



Contrail™

Contrail Feature Guide

Release

4.0



Modified: 2018-02-20

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

Juniper Networks, the Juniper Networks logo, Juniper, and Junos are registered trademarks of Juniper Networks, Inc. and/or its affiliates in the United States and other countries. All other trademarks may be property of their respective owners.

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

Contrail™ Contrail Feature Guide

4.0

Copyright © 2018 Juniper Networks, Inc. All rights reserved.

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

YEAR 2000 NOTICE

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

END USER LICENSE AGREEMENT

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

Table of Contents

	About the Documentation	xxiii
	Documentation and Release Notes	xxiii
	Documentation Conventions	xxiii
	Documentation Feedback	xxv
	Requesting Technical Support	xxvi
	Self-Help Online Tools and Resources	xxvi
	Opening a Case with JTAC	xxvi
Part 1	Overview	
Chapter 1	Understanding Contrail Controller	3
	Contrail Overview	3
	Contrail Description	4
	Contrail Major Components	4
	Contrail Control Nodes	4
	Contrail Compute Nodes – XMPP Agent and vRouter	4
	Contrail Solution	4
Part 2	Installing and Upgrading Contrail	
Chapter 2	Supported Platforms and Server Requirements	9
	Supported Platforms Contrail 4.0.x	9
	Server Requirements	10
Chapter 3	Installing Contrail and Provisioning Roles	11
	Introduction to Containerized Contrail Modules	11
	Why Use Containers?	11
	Overview of Contrail Containers	12
	Contrail 4.0 Containers	13
	contrail-controller	13
	contrail-analytics	14
	contrail-analyticsdb	14
	contrail-lb	14
	Summary of Container Design, Configuration Management, and Orchestration	14
	Downloading Installation Software	15
	Installing the Operating System and Contrail Packages	15
	Installing Containerized Contrail Clusters Using Server Manager	17
	Installing Server Manager	17
	Creating Objects with Server Manager and JSONs	18
	Preparing the Target System for Provisioning	19
	Provisioning the System	19

Installing Containerized Contrail Using Server Manager Lite (SM-Lite)	20
Preparing for SM-Lite Installation	20
Prerequisites	21
Installing SM-Lite	21
Provisioning Contrail Using SM-Lite	22
Sample JSONs and Testbed.py	22
Supporting Multiple Interfaces on Servers and Nodes	23
Support for Multiple Interfaces	23
Number of cfgm Nodes Supported	24
Uneven Number of Database Nodes Required	24
Support for VLAN Interfaces	24
Support for Bonding Options	24
Support for Static Route Options	24
Server Interface Examples	25
Interface Naming and Configuration Management	25
Configuring the Control Node	27
Adding a New Node to an Existing Containerized Contrail Cluster	31
Controller Configuration	32
Using contrailctl to add node configuration on existing containers	32
Manually configure contrailctl on all containers and sync the configs	33
Using contrailctl to Configure Services Within Containers	34
What is contrailctl?	34
Command Operations	34
Updating and Syncing Service Configurations Within the Container	35
Supporting Multiple Interfaces on Servers and Nodes	36
Support for Multiple Interfaces	36
Number of cfgm Nodes Supported	36
Uneven Number of Database Nodes Required	37
Support for VLAN Interfaces	37
Support for Bonding Options	37
Support for Static Route Options	37
Server Interface Examples	38
Interface Naming and Configuration Management	38
Contrail Global Controller	39
Resource Identifier Management	39
Multiple Location Resource Provisioning	39
Requirements, Assumptions, and Constraints	40
Platform Support	40
Installation	40
Role- and Resource-Based Access Control	40
Contrail Role- and Resource-Based Access (RBAC) Overview	41
API-Level Access Control	41
Rule Sets and ACL Objects	42
Object Level Access Control	42
Configuration	42
Parameter: aaa-mode	43
Parameter: cloud_admin_role	43
Global Read-Only Role	44

	Parameter Changes in /etc/neutron/api-paste.ini	44
	Utilities	44
	Utility: rbacutil.py	45
	Utility: chmod2.py	45
	Upgrading from Previous Releases	45
	Configuring RBAC Using the Contrail User Interface	46
	Configuring RBAC at the Global Level	46
	Configuring RBAC at the Domain Level	46
	Configuring RBAC at the Project Level	47
	Configuring RBAC Details	47
	RBAC Resources	48
Chapter 4	Installation and Configuration Scenarios	49
	Setting Up and Using a Simple Virtual Gateway with Contrail 4.0	49
	Introduction to the Simple Gateway	49
	How the Simple Gateway Works	50
	Setup Without Simple Gateway	50
	Setup With a Simple Gateway	51
	Simple Gateway Configuration Features	52
	Packet Flows with the Simple Gateway	53
	Packet Flow Process From the Virtual Network to the Public Network	53
	Packet Flow Process From the Public Network to the Virtual Network	54
	Methods for Configuring the Simple Gateway	55
	Using the vRouter Configuration File to Configure the Simple Gateway	55
	Using Thrift Messages to Dynamically Configure the Simple Gateway	55
	How to Dynamically Create a Virtual Gateway	56
	How to Dynamically Delete a Virtual Gateway	56
	Using Devstack to Configure the Simple Gateway	57
	Common Issues with Simple Gateway Configuration	58
	Configuring MD5 Authentication for BGP Sessions	59
	Configuring the Data Plane Development Kit (DPDK) Integrated with Contrail	
	vRouter	60
	DPDK Support in Contrail	60
	Preparing the server.json File for Provisioning a Contrail 4.0 Cluster Node	
	with DPDK	60
	Support for Multiple UIO Drivers	61
	Creating a Flavor for DPDK in OpenStack Kilo	62
	Configuring Single Root I/O Virtualization (SR-IOV)	62
	Overview: Configuring SR-IOV	63
	Configuring SR-IOV Using JSON	63
	Preparing the testbed.py File for Provisioning a Contrail Cluster with	
	SR-IOV	63
	Enabling ASPM in BIOS	64
	Configuring SR-IOV Features Without env.sriov in testbed.py	64

	Launching SR-IOV Virtual Machines	65
	Using the Contrail UI to Enable and Launch an SR-IOV Virtual Machine	65
	Using the CLI to Enable and Launch SR-IOV Virtual Machines	66
	Configuring Virtual Networks for Hub-and-Spoke Topology	67
	Route Targets for Virtual Networks in Hub-and-Spoke Topology	67
	Example: Configuring Hub-and-Spoke Virtual Networks	68
	Troubleshooting Hub-and-Spoke Topology	68
	Configuring Transport Layer Security-Based XMPP in Contrail	72
	Overview: TLS-Based XMPP	72
	TLS XMPP in Contrail	72
	Configuring XMPP Client and Server in Contrail	72
	Configuring Control Node for XMPP Server	73
	Configuring DNS Server for XMPP Server	73
	Configuring Control Node for XMPP Client	73
	Configuring Graceful Restart and Long-lived Graceful Restart	74
	Application of Graceful Restart and Long-lived Graceful Restart	74
	BGP Graceful Restart Helper Mode	75
	Feature Highlights	75
	XMPP Helper Mode	75
	Configuration Parameters	76
	Cautions for Graceful Restart	77
	Configuring Graceful Restart with the Contrail User Interface	78
Chapter 5	Using Contrail with Kubernetes	79
	Contrail Integration with Kubernetes	79
	What is Kubernetes?	79
	What is a Kubernetes Pod?	80
	Configuration Modes for Contrail Integrated with Kubernetes	80
	Default Mode	80
	Namespace Isolation Mode	81
	Custom Isolation Mode	82
	Nested Mode	82
	Kubernetes Services	82
	Ingress	83
	Contrail Kubernetes Solution	83
	Contrail Kubernetes Manager	84
	ECMP Load-Balancers for Kubernetes Services	84
	HAProxy Loadbalancer for Kubernetes Ingress	84
	Security Groups for Kubernetes Network Policy	84
	Domain Name Server (DNS)	85
	Installing and Provisioning Containerized Contrail Controller for Kubernetes	85
	Installing and Provisioning Containerized Contrail Controller for Kubernetes	86
	inventory/my-inventory/hosts Inventory File	88
	Example: inventory/my-inventory/hosts File in a Stand-alone Contrail Kubernetes Cluster	89
	Example: Nested inventory/my-inventory/hosts File in a Nested Contrail Kubernetes Cluster	89

	inventory/my-inventory/group_vars/all.yml Inventory File	89
	Example: inventory/my-inventory/group_vars/all.yml File in a Stand-alone Contrail Kubernetes Cluster	91
	Example: inventory/my-inventory/group_vars/all.yml File in a Nested Contrail Kubernetes Cluster	92
	Viewing Configuration for CNI for Kubernetes	93
	View Pod Name and IP Address	93
	Verify Reachability of Pods	93
	Verify If Isolated Namespace-Pods Are Not Reachable	94
	Verify If Non-Isolated Namespace-Pods Are Reachable	94
	Verify If a Namespace is Isolated	95
Chapter 6	Using Contrail with VMware vCenter	97
	Installing and Provisioning VMware vCenter with Contrail	97
	Overview: Integrating Contrail with vCenter Server	97
	Different Modes of vCenter Integration with Contrail	98
	vCenter-Only Mode	98
	vCenter-as-Compute Mode	99
	Preparing the Installation Environment	100
	Installation for vCenter-Only Mode	100
	Installing the vCenter-Only Components	101
	Installation for vCenter-as-Compute Mode	102
	Installing the vCenter-as-Compute Components	104
	Verification	105
	Adding Hosts or Nodes	105
	Adding an ESXi Host to an Existing vCenter Cluster	105
	Adding a vCenter Cluster to vCenter-as-Compute	105
	Underlay Network Configuration for ContrailVM	106
	Standard Switch Setup	106
	Distributed Switch Setup	107
	PCI Pass-Through Setup	109
	SR-IOV Setup	110
	Sample Testbed.py Files for Contrail vCenter	114
	Sample Testbed.py File for vCenter-Only Mode	114
	Sample Testbed.py File for vCenter-as-Compute Mode	116
	Using the Contrail and VMWare vCenter User Interfaces to Manage the Network	119
	Overview: User Interfaces for Contrail Integration with VMware vCenter	119
	Contrail Administration User Interface	119
	Contrail vCenter User Interface	119
	Feature Configuration for Contrail vCenter	120
	Creating a Virtual Network	120
	Creating a Virtual Machine	127
	Create a Virtual Machine – vCenter UI	128
	Configuring the vCenter Network in Contrail UI	133

Chapter 7	Using Contrail with Red Hat	135
	Deploying Contrail with Red Hat OpenStack Platform Director 10	135
	Overview	135
	TripleO Features	135
	Composable Roles	136
	Deployment Tools	137
	Preparing the Environment for Deployment	137
	Preparing for the Contrail Roles	138
	Preparing for the Underlay Network	138
	Preparing for the Provisioning Network	139
	Network Isolation	139
	Deploying an OSPD-10 Overcloud	141
	Installing the Undercloud	141
	Configuring Undercloud and Overcloud	142
	Defining Nodes with Ironic	142
	Configuring the Overcloud	144
	Get Contrail Components	144
	Configure NICs for Overcloud Networking	145
	Assign Addresses and Credentials	146
	Deploying the Overcloud	147
	Sample instackenv.json	148
	Adding a New Physical Compute Node	153
	Sample Heat Templates for NICs	154
	Example 1: NIC-1 to control plane; NIC-2 bridged interface	154
	Example 2: NIC-1 to control plane; NIC-2 and NIC-3 bond interface;	
	NIC-4 other networks	159
	What are NIC Templates?	162
	How NIC Templates Work	162
	Common Topologies	163
	Contrail NIC Templates	164
	Resource Registry Example	164
	NIC Templates for Control Nodes	165
	NIC Templates for Compute Nodes	168
	NIC Compute Node Template Subsection: Definition for NIC1	170
	NIC Compute Node Template Subsection: Definition for NIC2	170
Chapter 8	Using Server Manager to Automate Provisioning	173
	Installing Server Manager	173
	Installation Requirements for Server Manager	173
	Platform Support	173
	Installation Prerequisites	174
	Installing Server Manager	174
	Finishing the Installation	175
	Starting the Server Manager Service	176
	Upgrading Server Manager Software	176
	Steps for Upgrading	176
	Server Manager Installation Completion Checks	176
	Server Manager Checks	176
	Server Manager Client Checks	177

Server Manager WebUI Checks	177
Sample Configurations for Server Manager Templates	177
Sample Settings	177
Sample dhcp.template File	177
Sample named.conf.options File	178
Sample named.template File	178
The sendmail.cf File	178
Using Server Manager to Automate Provisioning	178
Overview of Server Manager	179
Server Manager Requirements and Assumptions	179
Server Manager Component Interactions	181
Configuring Server Manager	182
Configuring the Cobbler DHCP Template	184
User-Defined Tags for Server Manager	184
Server Manager Client Configuration File	185
Restart Services	185
Accessing Server Manager	186
Communicating with the Server Manager Client	187
Server Manager Commands for Configuring Servers	187
Server Manager Commands Common Options	187
Add New Servers or Update Existing Servers	188
Delete Servers	189
Display Server Configuration	189
Server Manager Commands for Managing Clusters	190
Server Manager Commands for Managing Tags	191
Server Manager Commands for Managing Images	193
Server Manager Operational Commands for Managing Servers	196
Reimaging Server(s)	196
Provisioning and Configuring Roles on Servers	197
Restarting Server(s)	198
Show Status of Server(s)	199
Show Status of Provision	199
Server Manager REST API Calls	200
REST APIs for Server Manager Configuration Database Entries	200
API: Add a Server	200
API: Delete Servers	201
API: Retrieve Server Configuration	201
API: Add an Image	201
API: Upload an Image	202
API: Get Image Information	202
API: Delete an Image	202
API: Add or Modify a Cluster	203
API: Delete a Cluster	203
API: Get Cluster Configuration	203
API: Get All Server Manager Configurations	203
API: Reimage Servers	204
API: Provision Servers	204

API: Restart Servers	204
Example: Reimaging and Provisioning a Server	205
Using the Server Manager Web User Interface	206
Log In to Server Manager	207
Create a Cluster for Server Manager	207
Edit a Cluster through Edit JSON	216
Working with Servers in the Server Manager User Interface	216
Add a Server	217
Edit Tags for Servers	219
Using the Edit Config Option for Multiple Servers	219
Edit a Server through Server Manager, Edit JSON	220
Filter Servers by Tag	220
Viewing Server Details	221
Configuring Images and Packages	223
Add New Image or Package	223
Selecting Server Manager Actions for Clusters	224
Reimage a Cluster	224
Provision a Cluster	224
Installing and Using Server Manager Lite	225
Server Manager Lite Overview	225
Installing Server Manager Lite	225
Provisioning Using SM-Lite with Contrail 4.0	226
Displaying the Cluster Status	227
Displaying the SM-Lite Installation and Provisioning Log Files	227
Contrail Provisioning Log Files	227
Chapter 9	
Extending Contrail to Physical Routers, Bare Metal Servers, Switches, and Interfaces	229
Using ToR Switches and OVSDb to Extend the Contrail Cluster to Other Instances	229
Support for ToR Switch and OVSDb Overview	229
ToR Services Node (TSN)	230
Contrail ToR Agent	230
Configuration Model	230
Control Plane	231
Data Plane	232
Using the Web Interface to Configure ToR Switch and Interfaces	232
Configuration Parameters for Provisioning ToR and TSN	234
Inventory Format ToR and TSN	235
JSON Format ToR and TSN	235
Testbed.py Format ToR and TSN	236
Prerequisite Configuration for QFX5100 Series Switch	237
Debug QFX5100 Configuration	238
Changes to Agent Configuration File	238
REST APIs	239
Configuring High Availability for the Contrail OVSDb ToR Agent	239
Overview: High Availability for a ToR Switch	239
High Availability Solution for Contrail ToR Agent	240
Failover Methodology Description	241

Failure Scenarios	241
Redundancy for HAProxy	243
Configuration for ToR Agent High Availability	243
Using Device Manager to Manage Physical Routers	244
Support for Physical Routers Overview	245
Configuration Model	245
Configuring a Physical Router	246
Alternate Ways to Configure a Physical Router	247
Device Manager Configurations	247
Prerequisite Configuration Required on MX Series Device	248
Debugging Device Manager Configuration	248
Configuration Scenarios	248
Configuring Physical Routers Using REST APIs	248
Sample Python Script Using Rest API for Configuring an MX Device ..	248
Device Manager Functionality	248
Dynamic Tunnels	249
Web UI Configuration	249
BGP Groups	251
Extending the Private Network	253
Extending the Public Network	255
Ethernet VPN Configuration	256
Floating IP Addresses and Source Network Address Translation for Guest	
Virtual Machines and Bare Metal Servers	257
Samples of Generated Configurations for an MX Series Device	265
Scenario 1: Physical Router With No External Networks	265
Scenario 2: Physical Router With External Network, Public VRF	267
Scenario 3: Physical Router With External Network, Public VRF, and	
EVPN	268
Scenario 4: Physical Router With External Network, Public VRF, and	
Floating IP Addresses for a Bare Metal Server	269
SR-IOV VF as the Physical Interface of vRouter	269
Using Gateway Mode to Support Remote Instances	270
Gateway Mode Overview	271
Provisioning Gateway Mode	271
Configuring Gateway Mode	271
External Configuration	272
Configuring Gateway Mode and High Availability	272
Scaling	272
REST APIs for Extending the Contrail Cluster to Physical Routers, and Physical	
and Logical Interfaces	273
Introduction: REST APIs for Extending Contrail Cluster	273
REST API for Physical Routers	273
REST API for Physical Interfaces	274
REST API for Logical Interfaces	275

Chapter 10	Installing and Using Contrail Storage	279
	Installing and Using Contrail Storage	279
	Overview of the Contrail Storage Solution	279
	Basic Storage Functionality with Contrail	280
	Ceph Block and Object Storage Functionality	280
	Using the Contrail Storage User Interface	280
	Hardware Specifications	282
	Contrail Storage Provisioning	282
Chapter 11	Upgrading Contrail Software	285
	Upgrading Contrail 3.2 to 4.0	285
	Overview of Changes in Contrail Release 4.0	285
	Assumptions	286
	Using a Server Manager Script to Upgrade from Contrail 3.2 to 4.0	286
	Running the Upgrade Script	286
	Steps for Upgrading Contrail 3.2 to 4.0 (Without using Server-Manager for upgrade)	286
	Dynamic Kernel Module Support (DKMS) for vRouter	288
Part 3	Configuring Contrail	
Chapter 12	Configuring Virtual Networks	293
	Creating Projects in OpenStack for Configuring Tenants in Contrail	294
	Creating a Virtual Network with Juniper Networks Contrail	296
	Creating a Virtual Network with OpenStack Contrail	299
	Creating an Image for a Project in OpenStack Contrail	301
	Creating a Floating IP Address Pool	304
	Using Security Groups with Virtual Machines (Instances)	305
	Security Groups Overview	305
	Creating Security Groups and Adding Rules	305
	Support for IPv6 Networks in Contrail	308
	Overview: IPv6 Networks in Contrail	309
	Creating IPv6 Virtual Networks in Contrail	309
	Address Assignments	310
	Adding IPv6 Peers	311
	Configuring EVPN and VXLAN	311
	Configuring the VXLAN Identifier Mode	313
	Configuring Forwarding	315
	Configuring the VXLAN Identifier	316
	Configuring Encapsulation Methods	317
Chapter 13	Example of Deploying a Multi-Tier Web Application Using Contrail	321
	Example: Deploying a Multi-Tier Web Application	321
	Multi-Tier Web Application Overview	321
	Example: Setting Up Virtual Networks for a Simple Tiered Web Application	322
	Verifying the Multi-Tier Web Application	324
	Sample Addressing Scheme for Simple Tiered Web Application	325
	Sample Physical Topology for Simple Tiered Web Application	326

	Sample Physical Topology Addressing	326
	Sample Network Configuration for Devices for Simple Tiered Web Application	327
Chapter 14	Configuring Services	333
	Configuring DNS Servers	333
	DNS Overview	333
	Defining Multiple Virtual Domain Name Servers	334
	IPAM and Virtual DNS	334
	DNS Record Types	335
	Configuring DNS Using the Interface	336
	Configuring DNS Using Scripts	341
	Distributed Service Resource Allocation with Containerized Contrail	342
	Replacement of Centralized Discovery Service in Contrail 4.0	342
	New Distributed Resource Allocation Manager	343
	Deprecation of IF-MAP	343
	Changes in Configuration Files	343
	Support for Multicast	349
	Subnet Broadcast	350
	All-Broadcast/Limited-Broadcast and Link-Local Multicast	350
	Host Broadcast	351
	Using Static Routes with Services	351
	Static Routes for Service Instances	351
	Configuring Static Routes on a Service Instance	352
	Configuring Static Routes on Service Instance Interfaces	353
	Configuring Static Routes as Host Routes	355
	Configuring Metadata Service	355
Chapter 15	Configuring Service Chaining	357
	Service Chaining	357
	Service Chaining Basics	357
	Service Chaining Configuration Elements	359
	Service Chaining MX Series Configuration	361
	ECMP Load Balancing in the Service Chain	362
	Customized Hash Field Selection for ECMP Load Balancing	363
	Overview: Custom Hash Feature	363
	Using ECMP Hash Fields Selection	365
	Configuring ECMP Hash Fields Over Service Chains	365
	Sample Flows	366
	Sample Traffic Flow Path Without Custom ECMP Hash Fields	366
	Sample Traffic Flow Path With Custom ECMP Hash Fields	366
	Service Chain Version 2 with Port Tuple	367
	Overview of Port Tuple	367
	Service Chain Version 2 Sample Workflow	368
	Service Chain with Equal-Cost Multipath in Active-Active Mode	369
	Service Chain Active-Standby Mode with Allowed Address Pair	369
	Service Chain with Static Route Table	370
	Service Chain with Health Check	370

	Using the Contrail Heat Template	371
	Introduction to Heat	371
	Heat Architecture	371
	Support for Heat Version 2 Resources	371
	Deprecation of Heat Version 1 Resources	372
	Heat Version 2 with Service Chaining and Port Tuple Sample Workflow	372
	Example: Creating a Service Template Using Heat	373
	Service Chain Route Reorigination	374
	Overview: Service Chaining in Contrail	374
	Route Aggregation	376
	Schema for Route Aggregation	377
	Configuring and Troubleshooting Route Aggregation	378
	Routing Policy	382
	Applying Routing Policy	383
	Routing Policy Configuration	385
	Configuring and Troubleshooting Routing Policy	386
	Using a VNC Script to Create Routing Policy	387
	Verify Routing Policy in API Server	388
	Verify Routing Policy in the Control Node	389
	Verify Routing Policy Configuration in the Control Node	389
	Verify Routing Policy Configuration on the Routing Instance	389
	Control for Route Reorigination	390
	Configuring and Troubleshooting Reorigination Control	391
	Health Check Object	392
	Health Check Overview	392
	Health Check Object Configuration	392
	Health Check Modes	393
	Creating a Health Check with the Contrail User Interface	394
	Using the Health Check	395
	Health Check Process	395
Chapter 16	Examples: Configuring Service Chaining	397
	Example: Creating an In-Network or In-Network-NAT Service Chain	397
	Creating an In-Network or In-Network-NAT Service Chain	397
	Example: Creating a Transparent Service Chain	405
	Creating a Transparent Mode Service Chain	405
	Example: Creating a Service Chain With the CLI	409
	CLI for Creating a Service Chain	409
	CLI for Creating a Service Template	410
	CLI for Creating a Service Instance	410
	CLI for Creating a Service Policy	410
	Example: Creating a Service Chain with VSRX and In-Network or Routed Mode	411
Chapter 17	Adding Physical Network Functions in Service Chains	413
	Using Physical Network Functions in Contrail Service Chains	413
	PNF Service Chaining Objects	413
	Prerequisites and Assumptions	414
	Example: Adding a Physical Network Function Device to a Service Chain	414
	Prerequisites for Adding a PNF to a Service Chain	415

Chapter 18	Configuring High Availability	423
	Juniper OpenStack High Availability	423
	Introduction	423
	Contrail High Availability	424
	OpenStack High Availability	424
	Supported Platforms	424
	Juniper OpenStack High Availability Architecture	424
	Juniper OpenStack Objectives	425
	Limitations	425
	Solution Components	426
	Virtual IP with Load Balancing	426
	Failure Handling	426
	Deployment	427
	Minimum Hardware Requirement	427
	Compute	427
	Network	428
	Installation	428
	Sample Server Manager-JSON	429
	High Availability Support Options	430
	Contrail High Availability Features	430
	Configuration Options for Enabling Contrail High Availability	430
	Supported Cluster Topologies for High Availability	431
	Deploying OpenStack and Contrail on the Same Highly Available Nodes	431
	Deploying OpenStack and Contrail on Different High Available Nodes	431
	Deploying Contrail Only on High Available Nodes	432
	High Availability for Containerized Contrail	433
	Containers for High Availability	433
	How High Availability is Handled in Contrail Containers	433
	contrail-lb	433
	High Availability for contrail-controller Container	433
	contrail-analytics	434
	contrail-analyticsdb	434
Chapter 19	Multitenancy Support	435
	Configuring Multitenancy Support	435
	Multitenancy Permissions	435
	API Server	436
	API Library Keystone Integration	437
	Supporting Utilities	437
	Quality of Service in Contrail	438
	Overview: Quality of Service	438
	Contrail QoS Model	438
	Features of Fabric Interfaces	438
	QoS Configuration Parameters for Provisioning	439
	Testbed.py Parameters	439
	JSON Parameters	440
	Queuing Implementation	441
	QoS Features by Release	441
	Contrail QoS Configuration Objects	441

Example: Mapping Traffic to Forwarding Classes	443
QoS Configuration Object Marking on the Packet	444
Traffic Originated by a Virtual Machine Interface	444
Traffic Destined to a Virtual Machine Interface	444
Traffic from a vhost Interface	444
Traffic from fabric interface	444
QoS Configuration Priority by Level	444
Queuing	444
Queue Selection in Datapath	445
Hardware Queueing in Linux kernel based vRouter	445
Parameters for QoS Scheduling Configuration	445
Configuring Network QoS Parameters	446
Overview	446
QoS Configuration Examples	446
Limitations	447
BGP as a Service	448
Contrail BGPaaS Features	448
BGPaaS Customer Use Cases	449
Dynamic Tunnel Insertion Within a Tenant Overlay	449
Dynamic Network Reachability of Applications	449
Liveness Detection for High Availability	450
Configuring BGPaaS	450
Configuring BGPaaS Using VNC API	450
Using the Contrail User Interface to Configure BGPaaS	451
BGP as a Service in Contrail Release 3.1	451
Chapter 20 Load Balancers	453
Using Load Balancers in Contrail	453
Invoking LBaaS Drivers	453
Using a Service Appliance Set as the LBaaS Provider	455
Understanding the Load Balancer Agent	456
F5 Networks Load Balancer Integration in Contrail	456
F5 Load Balancer Global Routed Mode	457
Initial Configuration on an F5 Device	459
Initial Configuration on an MX Series Device Used as DC Gateway	459
Example: Creating a Load Balancer	460
Using the Avi Networks Load Balancer for Contrail	461
Installing the Avi LBaaS Neutron Driver	461
Installing the Avi LBaaS Contrail Driver	463
Configuring the Avi Controller	464
Support for OpenStack LBaaS Version 2.0 APIs	465
Platform Support	466
Using OpenStack LBaaS Version 2.0	466
Support for Multiple Certificates per Listener	466
Neutron Load-Balancer Creation	467
Configuring Load Balancing as a Service in Contrail	467
Overview: Load Balancing as a Service	467
Contrail LBaaS Implementation	469

Chapter 21	Optimizing Contrail	473
	Route Target Filtering	473
	Introduction	473
	Debugging and Troubleshooting Route Target Filtering	474
	RTF Limitations in Contrail 1.10	475
	Source Network Address Translation (SNAT)	475
	Overview	475
	Neutron APIs for Routers	476
	Network Namespace	477
	Using Web UI to Configure Routers with SNAT	477
	Multiqueue Virtio Interfaces in Virtual Machines	478
	Multiqueue Virtio Overview	478
	Requirements and Setup for Multiqueue Virtio Interfaces	478
	Setting Virtual Machine Metadata for Multiple Queues	479
	vRouter Command Line Utilities	479
	Overview	479
	vif Command	480
	flow Command	482
	vrfstats Command	484
	rt Command	484
	dropstats Command	485
	mpls Command	489
	mirror Command	491
	vxlan Command	492
	nh Command	493
Part 4	Monitoring and Troubleshooting Contrail	
Chapter 22	Configuring Traffic Mirroring to Monitor Network Traffic	499
	Configuring Traffic Analyzers and Packet Capture for Mirroring	499
	Traffic Analyzer Images	499
	Configuring Traffic Analyzers	500
	Setting Up Traffic Mirroring Using Monitor > Debug > Packet Capture	500
	Setting Up Traffic Mirroring Using Configure > Networking > Services	504
	Configuring Interface Monitoring and Mirroring	509
	Mirroring Enhancements	510
	Mirroring Specified Traffic	510
	Configuring Headers and Next Hops	510
	How Mirroring is Implemented	511
	Analyzer Service Virtual Machine	511
	Packet Format for Analyzer	511
	Metadata Format	512
	Wireshark Changes	513
	Troubleshooting Packet Display	513
	Mapping VLAN Tags from a Physical NIC to a VMI (NIC-Assisted Mirroring)	513

Chapter 23	Understanding Contrail Analytics	515
	Understanding Contrail Analytics	515
	Contrail Alerts	516
	Alert API Format	516
	Analytics APIs for Alerts	517
	Analytics APIs for SSE Streaming	518
	Built-in Node Alerts	518
	Underlay Overlay Mapping in Contrail	519
	Overview: Underlay Overlay Mapping using Contrail Analytics	520
	Underlay Overlay Analytics Available in Contrail	520
	Architecture and Data Collection	521
	New Processes/Services for Underlay Overlay Mapping	521
	External Interfaces Configuration for Underlay Overlay Mapping	522
	Physical Topology	522
	SNMP Configuration	523
	Link Layer Discovery Protocol (LLDP) Configuration	523
	IPFIX and sFlow Configuration	523
	Sending pRouter Information to the SNMP Collector in Contrail	525
	pRouter UVEs	525
	Contrail User Interface for Underlay Overlay Analytics	527
	Enabling Physical Topology on the Web UI	527
	Viewing Topology to the Virtual Machine Level	528
	Viewing the Traffic of any Link	528
	Trace Flows	529
	Search Flows and Map Flows	530
	Overlay to Underlay Flow Map Schemas	530
	Module Operations for Overlay Underlay Mapping	533
	SNMP Collector Operation	533
	Topology Module Operation	534
	IPFIX and sFlow Collector Operation	535
	Troubleshooting Underlay Overlay Mapping	535
	Script to add pRouter Objects	536
Chapter 24	Configuring Contrail Analytics	539
	Analytics Scalability	540
	High Availability for Analytics	541
	System Log Receiver in Contrail Analytics	542
	Overview	542
	Redirecting System Logs to Contrail Collector	542
	Exporting Logs from Contrail Analytics	542
	Sending Flow Messages to the Contrail System Log	543
	Ceilometer Support in a Contrail Cloud	544
	Overview	544
	Ceilometer Details	544
	Verification of Ceilometer Operation	545
	Contrail Ceilometer Plugin	547
	Ceilometer Installation and Provisioning	549

	User Configuration for Analytics Alarms and Log Statistics	549
	Configuring Alarms Based on User-Visible Entities Data	549
	Examples: Detecting Anomalies	551
	Configuring the User-Defined Log Statistic	552
	Implementing the User-Defined Log Statistic	555
	Alarms History	557
	Viewing Alarms History	557
	Node Memory and CPU Information	558
	Role- and Resource-Based Access Control for the Contrail Analytics API	559
	Configuring Analytics as a Standalone Solution	560
	Overview: Contrail Analytics as a Standalone Solution	560
	Configuration Examples for Standalone	560
	Examples: Inventory File Controller Components	560
	JSON Configuration Examples	561
	Configuring Secure Sandesh and Introspect for Contrail Analytics	562
	Configuring Secure Sandesh Connection	562
	Configuring Secure Introspect Connection	563
Chapter 25	Using Contrail Analytics to Monitor and Troubleshoot the Network	565
	Monitoring the System	566
	Debugging Processes Using the Contrail Introspect Feature	568
	Monitor > Infrastructure > Dashboard	572
	Monitor Dashboard	572
	Monitor Individual Details from the Dashboard	573
	Using Bubble Charts	573
	Color-Coding of Bubble Charts	574
	Monitor > Infrastructure > Control Nodes	574
	Monitor Control Nodes Summary	575
	Monitor Individual Control Node Details	575
	Monitor Individual Control Node Console	577
	Monitor Individual Control Node Peers	579
	Monitor Individual Control Node Routes	580
	Monitor > Infrastructure > Virtual Routers	581
	Monitor vRouters Summary	582
	Monitor Individual vRouters Tabs	583
	Monitor Individual vRouter Details Tab	583
	Monitor Individual vRouters Interfaces Tab	584
	Monitor Individual vRouters Networks Tab	585
	Monitor Individual vRouters ACL Tab	586
	Monitor Individual vRouters Flows Tab	587
	Monitor Individual vRouters Routes Tab	588
	Monitor Individual vRouter Console Tab	589
	Monitor > Infrastructure > Analytics Nodes	591
	Monitor Analytics Nodes	591
	Monitor Analytics Individual Node Details Tab	592
	Monitor Analytics Individual Node Generators Tab	593
	Monitor Analytics Individual Node QE Queries Tab	594
	Monitor Analytics Individual Node Console Tab	595

Monitor > Infrastructure > Config Nodes	596
Monitor Config Nodes	596
Monitor Individual Config Node Details	597
Monitor Individual Config Node Console	598
Monitor > Networking	599
Monitor > Networking Menu Options	599
Monitor -> Networking -> Dashboard	600
Monitor > Networking > Projects	601
Monitor Projects Detail	602
Monitor > Networking > Networks	604
Query > Flows	607
Query > Flows > Flow Series	607
Example: Query Flow Series	610
Query > Flow Records	611
Query > Flows > Query Queue	613
Query > Logs	614
Query > Logs Menu Options	615
Query > Logs > System Logs	615
Sample Query for System Logs	616
Query > Logs > Object Logs	617
Understanding Flow Sampling	619
Flow Sampling	619
Flow Handling	620
Flow Aging	620
TCP State-Based Flow Handling and Aging	621
TCP State-Based Flow Handling	621
Protocol Based Flow Aging	621
Fat Flow	622
Example: Debugging Connectivity Using Monitoring for Troubleshooting	622
Using Monitoring to Debug Connectivity	622
Chapter 26 Common Support Answers	629
Debugging Ping Failures for Policy-Connected Networks	629
Debugging BGP Peering and Route Exchange in Contrail	635
Example Cluster	636
Verifying the BGP Routers	636
Verifying the Route Exchange	639
Debugging Route Exchange with Policies	641
Debugging Peering with an MX Series Router	643
Debugging a BGP Peer Down Error with Incorrect Family	645
Configuring MX Peering (iBGP)	647
Checking Route Exchange with an MX Series Peer	649
Checking the Route in the MX Series Router	650
Troubleshooting the Floating IP Address Pool in Contrail	651
Example Cluster	652
Example	653
Example: MX80 Configuration for the Gateway	654
Ping the Floating IP from the Public Network	656
Troubleshooting Details	656

	Get the UUID of the Virtual Network	656
	View the Floating IP Object in the API Server	657
	View floating-ips in floating-ip-pools in the API Server	660
	Check Floating IP Objects in the Virtual Machine Interface	662
	View the BGP Peer Status on the Control Node	665
	Querying Routes in the Public Virtual Network	666
	Verification from the MX80 Gateway	667
	Viewing the Compute Node Vnsw Agent	669
	Advanced Troubleshooting	672
	Removing Stale Virtual Machines and Virtual Machine Interfaces	674
	Problem Example	674
	Show Virtual Machines	675
	Show Virtual Machines Using Python API	677
	Delete Methods	678
	Troubleshooting Link-Local Services in Contrail	678
	Overview of Link-Local Services	678
	Troubleshooting Procedure for Link-Local Services	678
	Metadata Service	681
	Troubleshooting Procedure for Link-Local Metadata Service	681
Part 5	Contrail Commands and APIs	
Chapter 27	Contrail Commands	685
	Getting Contrail Node Status	685
	Overview	685
	UVE for NodeStatus	685
	Node Status Features	686
	Using Introspect to Get Process Status	691
	contrail-status script	692
	contrail-logs (Accessing Log File Messages)	694
	Command-Line Options for Contrail-Logs	694
	Option Descriptions	694
	Example Uses	695
	contrail-status (Viewing Node Status)	697
	contrail-version (Viewing Version Information)	699
	service (Managing Services)	701
	Backing Up Contrail Databases Using JSON Format	702
	Preliminary Cautions	702
	Simple Backup Using JSON Format	702
	Restore Simple Database Backup	703
	Example Backup and Restore With JSON	704
	Example: Perform Simple Backup	704
	Example: Perform Restore	705

Chapter 28	Contrail Application Programming Interfaces (APIs)	709
	Contrail Analytics Application Programming Interfaces (APIs) and User-Visible	
	Entities (UEs)	709
	User-Visible Entities	710
	Common UEs in Contrail	711
	Virtual Network UVE	711
	Virtual Machine UVE	711
	vRouter UVE	712
	UEs for Contrail Nodes	712
	Wild Card Query of UEs	713
	Filtering UVE Information	713
	Log and Flow Information APIs	719
	HTTP GET APIs	720
	HTTP POST API	720
	POST Data Format Example	721
	Query Types	722
	Examining Query Status	722
	Examining Query Chunks	722
	Example Queries for Log and Flow Data	723
	Working with Neutron	726
	Data Structure	726
	Network Sharing in Neutron	726
	Commands for Neutron Network Sharing	727
	Support for Neutron APIs	727
	Contrail Neutron Plugin	728
	DHCP Options	728
	Incompatibilities	729
	Support for Amazon VPC APIs on Contrail OpenStack	729
	Overview of Amazon Virtual Private Cloud	729
	Mapping Amazon VPC Features to OpenStack Contrail Features	730
	VPC and Subnets Example	730
	Euca2ools CLI for VPC and Subnets	731
	Security in VPC: Network ACLs Example	732
	Euca2ools CLI for Network ACLs	733
	Security in VPC: Security Groups Example	733
	Euca2ools CLI for Security Groups	734
	Elastic IPs in VPC	734
	Euca2ools CLI for Elastic IPs	735
	Euca2ools CLI for Route Tables	735
	Supported Next Hops	735
	Internet Gateway Next Hop Euca2ools CLI	736
	NAT Instance Next Hop Euca2ools CLI	736
	Example: Creating a NAT Instance with Euca2ools CLI	736

About the Documentation

- Documentation and Release Notes on page xxiii
- Documentation Conventions on page xxiii
- Documentation Feedback on page xxv
- Requesting Technical Support on page xxvi

Documentation and Release Notes

To obtain the most current version of all Juniper Networks® technical documentation, see the product documentation page on the Juniper Networks website at <http://www.juniper.net/techpubs/>.

If the information in the latest release notes differs from the information in the documentation, follow the product Release Notes.

Juniper Networks Books publishes books by Juniper Networks engineers and subject matter experts. These books go beyond the technical documentation to explore the nuances of network architecture, deployment, and administration. The current list can be viewed at <http://www.juniper.net/books>.

Documentation Conventions

Table 1 on page xxiv defines notice icons used in this guide.

Table 1: Notice Icons







Icon	Meaning	Description
	Informational note	Indicates important features or instructions.
	Caution	Indicates a situation that might result in loss of data or hardware damage.
	Warning	Alerts you to the risk of personal injury or death.
	Laser warning	Alerts you to the risk of personal injury from a laser.
	Tip	Indicates helpful information.
	Best practice	Alerts you to a recommended use or implementation.

Table 2 on page xxiv defines the text and syntax conventions used in this guide.

Table 2: Text and Syntax Conventions

Convention	Description	Examples
Bold text like this	Represents text that you type.	To enter configuration mode, type the configure command: user@host> configure
Fixed-width text like this	Represents output that appears on the terminal screen.	user@host> show chassis alarms No alarms currently active
<i>Italic text like this</i>	<ul style="list-style-type: none"> Introduces or emphasizes important new terms. Identifies guide names. Identifies RFC and Internet draft titles. 	<ul style="list-style-type: none"> A policy <i>term</i> is a named structure that defines match conditions and actions. <i>Junos OS CLI User Guide</i> RFC 1997, <i>BGP Communities Attribute</i>
<i>Italic text like this</i>	Represents variables (options for which you substitute a value) in commands or configuration statements.	Configure the machine's domain name: [edit] root@# set system domain-name <i>domain-name</i>

Table 2: Text and Syntax Conventions (*continued*)

Convention	Description	Examples
Text like this	Represents names of configuration statements, commands, files, and directories; configuration hierarchy levels; or labels on routing platform components.	<ul style="list-style-type: none">To configure a stub area, include the stub statement at the [edit protocols ospf area area-id] hierarchy level.The console port is labeled CONSOLE.
< > (angle brackets)	Encloses optional keywords or variables.	stub <default-metric <i>metric</i>>;
(pipe symbol)	Indicates a choice between the mutually exclusive keywords or variables on either side of the symbol. The set of choices is often enclosed in parentheses for clarity.	broadcast multicast (<i>string1</i> <i>string2</i> <i>string3</i>)
# (pound sign)	Indicates a comment specified on the same line as the configuration statement to which it applies.	rsvp { # Required for dynamic MPLS only
[] (square brackets)	Encloses a variable for which you can substitute one or more values.	community name members [<i>community-ids</i>]
Indentation and braces ({ })	Identifies a level in the configuration hierarchy.	<pre>[edit] routing-options { static { route default { nexthop <i>address</i>; retain; } } }</pre>
;(semicolon)	Identifies a leaf statement at a configuration hierarchy level.	
GUI Conventions		
Bold text like this	Represents graphical user interface (GUI) items you click or select.	<ul style="list-style-type: none">In the Logical Interfaces box, select All Interfaces.To cancel the configuration, click Cancel.
> (bold right angle bracket)	Separates levels in a hierarchy of menu selections.	In the configuration editor hierarchy, select Protocols>Ospf .

Documentation Feedback

We encourage you to provide feedback, comments, and suggestions so that we can improve the documentation. You can provide feedback by using either of the following methods:

- Online feedback rating system—On any page of the Juniper Networks TechLibrary site at <http://www.juniper.net/techpubs/index.html>, simply click the stars to rate the content, and use the pop-up form to provide us with information about your experience. Alternately, you can use the online feedback form at <http://www.juniper.net/techpubs/feedback/>.

- E-mail—Send your comments to techpubs-comments@juniper.net. Include the document or topic name, URL or page number, and software version (if applicable).

Requesting Technical Support

Technical product support is available through the Juniper Networks Technical Assistance Center (JTAC). If you are a customer with an active J-Care or Partner Support Service support contract, or are covered under warranty, and need post-sales technical support, you can access our tools and resources online or open a case with JTAC.

- JTAC policies—For a complete understanding of our JTAC procedures and policies, review the *JTAC User Guide* located at <http://www.juniper.net/us/en/local/pdf/resource-guides/7100059-en.pdf>.
- Product warranties—For product warranty information, visit <http://www.juniper.net/support/warranty/>.
- JTAC hours of operation—The JTAC centers have resources available 24 hours a day, 7 days a week, 365 days a year.

Self-Help Online Tools and Resources

For quick and easy problem resolution, Juniper Networks has designed an online self-service portal called the Customer Support Center (CSC) that provides you with the following features:

- Find CSC offerings: <http://www.juniper.net/customers/support/>
- Search for known bugs: <https://prsearch.juniper.net/>
- Find product documentation: <http://www.juniper.net/documentation/>
- Find solutions and answer questions using our Knowledge Base: <http://kb.juniper.net/>
- Download the latest versions of software and review release notes: <http://www.juniper.net/customers/csc/software/>
- Search technical bulletins for relevant hardware and software notifications: <http://kb.juniper.net/InfoCenter/>
- Join and participate in the Juniper Networks Community Forum: <http://www.juniper.net/company/communities/>
- Open a case online in the CSC Case Management tool: <http://www.juniper.net/cm/>

To verify service entitlement by product serial number, use our Serial Number Entitlement (SNE) Tool: <https://entitlementsearch.juniper.net/entitlementsearch/>

Opening a Case with JTAC

You can open a case with JTAC on the Web or by telephone.

- Use the Case Management tool in the CSC at <http://www.juniper.net/cm/>.
- Call 1-888-314-JTAC (1-888-314-5822 toll-free in the USA, Canada, and Mexico).

For international or direct-dial options in countries without toll-free numbers, see <http://www.juniper.net/support/requesting-support.html>.

PART 1

Overview

- [Understanding Contrail Controller on page 3](#)

CHAPTER 1

Understanding Contrail Controller

- [Contrail Overview on page 3](#)
- [Contrail Description on page 4](#)

Contrail Overview

Juniper Networks Contrail is an open, standards-based software solution that delivers network virtualization and service automation for federated cloud networks. It provides self-service provisioning, improves network troubleshooting and diagnostics, and enables service chaining for dynamic application environments across enterprise virtual private cloud (VPC), managed Infrastructure as a Service (IaaS), and Networks Functions Virtualization use cases.

Contrail simplifies the creation and management of virtual networks to enable policy-based automation, greatly reducing the need for physical and operational infrastructure typically required to support network management. In addition, it uses mature technologies to address key challenges of large-scale managed environments, including multitenancy, network segmentation, network access control, and IP service enablement. These challenges are particularly difficult in evolving dynamic application environments such as the Web, gaming, big data, cloud, and the like.

Contrail allows a tenant or a cloud service provider to abstract virtual networks at a higher layer to eliminate device-level configuration and easily control and manage policies for tenant virtual networks. A browser-based user interface enables users to define virtual network and network service policies, then configure and interconnect networks simply by attaching policies. Contrail also extends native IP capabilities to the hosts (compute nodes) in the data center to address the scale, resiliency, and service enablement challenges of traditional orchestration platforms.

Using Contrail, a tenant can define, manage, and control the connectivity, services, and security policies of the virtual network. The tenant or other users can use the self-service graphical user interface to easily create virtual network nodes, add and remove IP services (such as firewall, load balancing, DNS, and the like) to their virtual networks, then connect the networks using traffic policies that are simple to create and apply. Once created, policies can be applied across multiple network nodes, changed, added, and deleted, all from a simple browser-based interface.

Contrail can be used with open cloud orchestration systems such as OpenStack. It can also interact with other systems and applications based on Operations Support System

(OSS) and Business Support Systems (BSS), using northbound APIs. Contrail allows customers to build elastic architectures that leverage the benefits of cloud computing — agility, self-service, efficiency, and flexibility — while providing an interoperable, scale-out control plane for network services within and across network domains.

Related Documentation

- [Contrail Description on page 4](#)

Contrail Description

- [Contrail Major Components on page 4](#)
- [Contrail Solution on page 4](#)

Contrail Major Components

The following are the major components of Contrail.

Contrail Control Nodes

- Responsible for the routing control plane, configuration management, analytics, and the user interface.
- Provide APIs to integrate with an orchestration system or a custom user interface.
- Horizontally scalable, can run on multiple servers.

Contrail Compute Nodes – XMPP Agent and vRouter

- Responsible for managing the data plane.
- Functionality can reside on a host OS.

Contrail Solution

Contrail architecture takes advantage of the economics of cloud computing and simplifies the physical network (IP fabric) with a software virtual network overlay that delivers service orchestration, automation, and intercloud federation for public and hybrid clouds.

Similar to the native Layer 3 designs of web-scale players in the market and public cloud providers, the Contrail solution leverages IP as the abstraction between dynamic applications and networks, ensuring smooth migration from existing technologies, as well as support of emerging dynamic applications.

The Contrail solution is software running on x86 Linux servers, focused on enabling multitenancy for enterprise Information Technology as a Service (ITaaS). Multitenancy is enabled by the creation of multiple distinct Layer 3-enabled virtual networks with traffic isolation, routing between tenant groups, and network-based access control for each user group. To extend the IP network edge to the hosts and accommodate virtual machine workload mobility while simplifying and automating network (re)configuration, Contrail maintains a real-time state across dynamic virtual networks, exposes the network-as-a-service to cloud users, and enables deep network diagnostics and analytics down to the host.

In this paradigm, users of cloud-based services can take advantage of services and applications and assume that pooled, elastic resources are orchestrated, automated, and optimized across compute, storage, and network nodes in a converged architecture that is application-aware and independent of underlying hardware and software technologies.

- Related Documentation**
- [Contrail Overview on page 3](#)
 - *Contrail Roles Overview*

PART 2

Installing and Upgrading Contrail

- [Supported Platforms and Server Requirements on page 9](#)
- [Installing Contrail and Provisioning Roles on page 11](#)
- [Installation and Configuration Scenarios on page 49](#)
- [Using Contrail with Kubernetes on page 79](#)
- [Using Contrail with VMware vCenter on page 97](#)
- [Using Contrail with Red Hat on page 135](#)
- [Using Server Manager to Automate Provisioning on page 173](#)
- [Extending Contrail to Physical Routers, Bare Metal Servers, Switches, and Interfaces on page 229](#)
- [Installing and Using Contrail Storage on page 279](#)
- [Upgrading Contrail Software on page 285](#)

CHAPTER 2

Supported Platforms and Server Requirements

- [Supported Platforms Contrail 4.0.x on page 9](#)
- [Server Requirements on page 10](#)

Supported Platforms Contrail 4.0.x

[Table 3 on page 9](#) lists the operating system versions and the corresponding Linux or Ubuntu kernel versions supported by Contrail Release 4.0.

Table 3: Supported Platforms

Contrail Release	OpenStack Release	Operating System and Kernel Versions
Contrail Release 4.0.1	OpenStack Ocata	<ul style="list-style-type: none">• Ubuntu 16.04.2—Linux kernel version 4.4.0-62-generic• VMware vCenter 6.0, 6.5—Ubuntu 16.04.2 kernel version 4.4.0-62-generic
	OpenStack Newton	<ul style="list-style-type: none">• Ubuntu 16.04.2—Linux kernel version 4.4.0-62-generic• VMware vCenter 6.0, 6.5—Ubuntu 16.04.2 kernel version 4.4.0-62-generic
	OpenStack Mitaka	<ul style="list-style-type: none">• Ubuntu 14.04.5—Linux kernel versions 3.13.0-110-generic and 4.4.0-34-generic• VMware vCenter 6.0, 6.5—Ubuntu 14.04.4 kernel version 3.13.0-110-generic
Contrail Release 4.0	OpenStack Newton	<ul style="list-style-type: none">• Ubuntu 16.04.2—Linux kernel version 4.4-62• Red Hat 7.3—Linux kernel version 3.10.0-514.6.2• VMware vCenter 5.5, 6.0—Ubuntu 14.04.4 kernel version 3.13.0-106-generic
	OpenStack Mitaka	<ul style="list-style-type: none">• Ubuntu 14.04.5—Linux kernel versions 3.13.0-106-generic and 4.4.0-34-generic• VMware vCenter 5.5, 6.0—Ubuntu 16.04.2 kernel version 4.4.0-62-generic



NOTE: In Contrail Release 4.0 and later, if the stock kernel version of your Ubuntu system is other than the required version, you can upgrade the kernel for all nodes in the cluster by using the following parameter in `cluster.json` for Server Manager or SM-Lite provisioning or `testbed.py`.

```
{
  "cluster" : [{
    "parameters" : {
      "provisioning" : {
        "contrail" : {
          "kernel_upgrade" : true
        }
      }
    }
  ]
}
```

Server Requirements

The minimum requirement for a proof-of-concept (POC) system is 3 servers, either physical or virtual machines. All non-compute roles can be configured in each controller node. For scalability and availability reasons, it is highly recommended to use physical servers.

Each server must have a minimum of:

- 64 GB memory
- 300 GB hard drive
- 4 CPU cores
- At least one Ethernet port



NOTE: If you are using Contrail Storage, additional hardware requirements can be found in *Installing and Using Contrail Storage*, Hardware Specifications.

Related Documentation

- [Contrail Roles Overview](#)
- [Downloading Installation Software on page 15](#)

CHAPTER 3

Installing Contrail and Provisioning Roles

- [Introduction to Containerized Contrail Modules on page 11](#)
- [Downloading Installation Software on page 15](#)
- [Installing the Operating System and Contrail Packages on page 15](#)
- [Installing Containerized Contrail Clusters Using Server Manager on page 17](#)
- [Installing Containerized Contrail Using Server Manager Lite \(SM-Lite\) on page 20](#)
- [Supporting Multiple Interfaces on Servers and Nodes on page 23](#)
- [Configuring the Control Node on page 27](#)
- [Adding a New Node to an Existing Containerized Contrail Cluster on page 31](#)
- [Using contrailctl to Configure Services Within Containers on page 34](#)
- [Supporting Multiple Interfaces on Servers and Nodes on page 36](#)
- [Contrail Global Controller on page 39](#)
- [Role- and Resource-Based Access Control on page 40](#)

Introduction to Containerized Contrail Modules

Starting with Contrail 4.0, some subsystems of Contrail are delivered as Docker containers.

- [Why Use Containers? on page 11](#)
- [Overview of Contrail Containers on page 12](#)
- [Contrail 4.0 Containers on page 13](#)
- [Summary of Container Design, Configuration Management, and Orchestration on page 14](#)

Why Use Containers?

Contrail software releases are distributed as sets of packages for each of the subsystem modules of a Contrail system. The Contrail modules depend on numerous open source packages and provisioning tools and are validated on specific Linux distributions. Each module has its own dependency chains and its own configuration parameters.

These dependencies lead to complexities of deployment, including:

- The Linux version of the target system must match exactly to the version upon which Contrail is qualified, or the installation might fail.
- A deployment that succeeds despite an operating system mismatch could pull dependent packages from a customer mirror site that don't match the dependencies with which the Contrail system was qualified, creating potential for failure.
- Change in any package on the target system creates a risk of failure of dependencies in the Contrail software, creating a need for requalification upon any system change.
- Currently, provisioning tools such as Fuel, Juju, Puppet, and the like interact directly with Contrail services. Over time, these tools become more complex, requiring interaction with the lowest level of details of Contrail service parameters.

Containerizing some Contrail subsystems reduces the complexity of deploying Contrail and provides a straightforward, simple way to deploy and operate Contrail.

Overview of Contrail Containers

Starting with Contrail 4.0, some of the Contrail subsystems are delivered as Docker containers that group together related functional components. Each container file includes an INI-based configuration file for configuring the services within the container. The purpose of the INI is to provide enough high-level configuration entries to configure all services within the container, while masking the complexity of the internal service configuration. The container configuration files are available on the host system and mounted within specific containers.

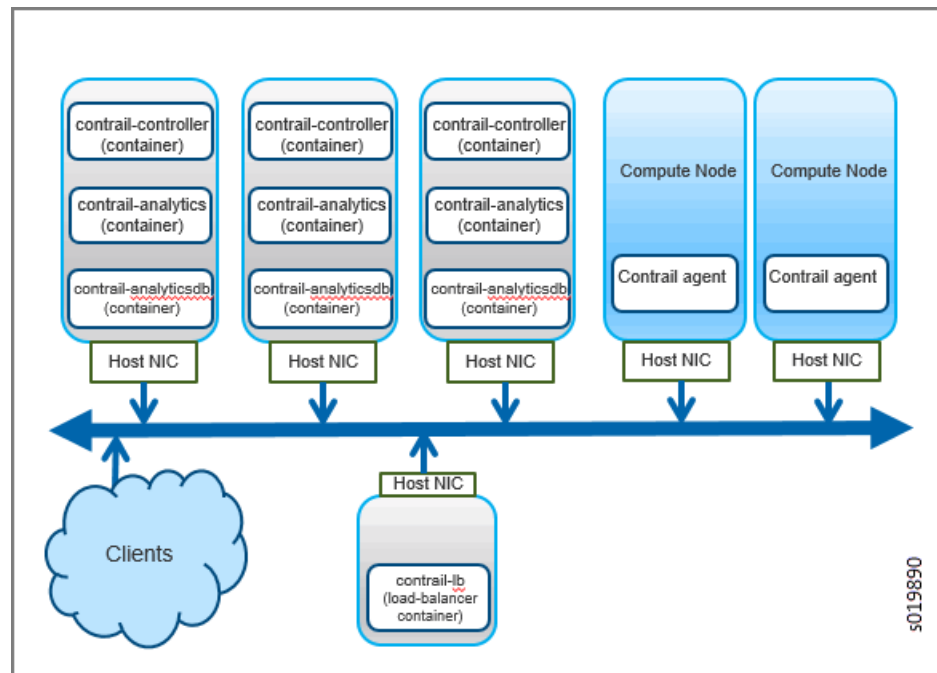
In Contrail 4.0, the containerized components include Contrail controller, analytics, and load-balancer applications. Contrail OpenStack components are not containerized at this time.

In Contrail 4.0.1, the containerized components include OpenStack Ocata services. Only OpenStack Ocata services are containerized. Mitaka and Newton SKUs of OpenStack are still provisioned as non-containerized host services.

All Contrail containers run with the host network, without using a Docker bridge, however, all services within the container listen on the host network interface. Some services, such as RabbitMQ, require extra parameters, such as a host-based PID namespace.

The intention is to build a composable Contrail core system of containers that can be used with differing cloud and container orchestration systems, such as OpenStack, Kubernetes, Mesos, and the like.

Figure 1: Sample Configuration Containerized Contrail



Contrail 4.0 Containers

This section describes the containers in Contrail 4.0 and their contents.

- [contrail-controller on page 13](#)
- [contrail-analytics on page 14](#)
- [contrail-analyticsdb on page 14](#)
- [contrail-lb on page 14](#)

contrail-controller

The **contrail-controller** container includes all Contrail applications that make up a Contrail controller, including:

- All configuration services, such as **contrail api**, **config-nodemgr**, **device-manager**, **schema**, **svc-monitor**, and **CONFIGDB**.
- All control services, such as **contrail-control**, **control-nodemgr**, **contrail-dns**, and **contrail-named**.
- All Web UI services, such as **contrail-webui** and **contrail-webui-middleware**.
- Configuration database (Cassandra)
- Zookeeper
- RabbitMQ
- Redis for Web Ui

contrail-analytics

The **contrail-analytics** container includes all Contrail analytics services, including:

- **alarm-gen**
- **analytics-api**
- **analytics-nodemgr**
- **contrail-collector**
- **query-engine**
- **snmp-collector**
- **contrail-topology**

contrail-analyticsdb

The **contrail-analyticsdb** container has Cassandra for the analytics database and Kafka for streaming data.

contrail-lb

The **contrail-lb** loadbalancer container includes all components that provide load-balancing and high availability to the system, such as HAproxy, keepalive, and the like.

In previous releases of Contrail, HAproxy and keepalive were included in most services to load-balance Contrail service endpoints. Starting with Contrail 4.0, the load-balancers are taken out of the individual services and held instead in a dedicated **loadbalancer** container. An exception is HAproxy as part of the vrouter agent, which can be used to implement Load-Balancing as a Service (LBaaS).

The **loadbalancer** container is an optional container, and customers can choose to use their own load-balancing system.

Summary of Container Design, Configuration Management, and Orchestration

The following are key features of the new architecture of Contrail containers.

- All of the Contrail containers are multiprocess Docker containers.
- Each container has an INI-based configuration file that has the configurations for all of the applications running in that container.
- The user toolset **contrailctl** is used to manage the container configuration files.
- Each container is self-contained, with minimal external orchestration needs.
- A single tool, Ansible, is used for all levels of building, deploying, and provisioning the containers. The Ansible code for the Contrail system is named **contrail-ansible** and kept in a separate repository. The Contrail Ansible code is responsible for all aspects of Contrail container build, deployment, and basic container orchestration.

- Related Documentation**
- [Using contrailctl to Configure Services Within Containers on page 34](#)
 - [Installing Containerized Contrail Clusters Using Server Manager on page 17](#)

Downloading Installation Software

All components necessary for installing the Contrail Controller are available as:

- an **RPM** file (**contrail-install-packages-1.xx-xxx.el6.noarch.rpm**) that can be used to install the Contrail system on an appropriate CentOS operating system.
- a **Debian** file (**contrail-install-packages-1.xx-xxx~xxxxxx_all.deb**) that can be used to install the Contrail system on an appropriate Ubuntu operating system.

Versions are available for each Contrail release, for the supported Linux operating systems and versions, and for the supported versions of OpenStack.

All installation images can be downloaded from
<https://www.juniper.net/support/downloads/?p=contrail#sw>.

The Contrail image includes the following software:

- All dependent software packages needed to support installation and operation of OpenStack and Contrail
- Contrail Controller software – all components
- OpenStack release currently in use for Contrail

- Related Documentation**
- [Installing the Operating System and Contrail Packages](#)
 - [Installing Containerized Contrail Clusters Using Server Manager on page 17](#)
 - [Installing Containerized Contrail Using Server Manager Lite \(SM-Lite\) on page 20](#)
 - [Download Software](#)

Installing the Operating System and Contrail Packages

Install the stock CentOS or Ubuntu operating system image appropriate for your version of Contrail onto the server. See [Supported Platforms Contrail 4.0.x](#) or [Supported Platforms Contrail 4.1](#). Then install Contrail packages separately.

The following are general guidelines for installing the operating system and preparing to install Contrail.

1. Install a CentOS or Ubuntu minimal distribution as desired on all servers. Typically, for CentOS this is a basic ISO install; for Ubuntu, use a core server install, with only OpenSSH and no other packages. Follow the published operating system installation procedure for the selected operating system; refer to the website for the operating system.
2. Install Contrail Server Manager.
3. Create an image.json with the Ubuntu or CentOS image to be used to reimage the target server.

Sample JSON Snippet

```
{
    "image": [
        {
            "category": "image",
            "id": "ubuntu-14.04.04",
            "parameters": {
                "kickseed": "/etc/contrail_smgr/kickstarts/contrail-ubuntu_trusty.seed",
                "kickstart":
"/etc/contrail_smgr/kickstarts/contrail-ubuntu_trusty.ks"
            },
            "path": "/path/to/ubuntu-image.iso",
            "type": "ubuntu",
            "version": "14.04.04"
        }
    ]
}
```

4. Use Server Manager to add the image.json, to be used for reimagining.

server-manager add image -f image.json

For full installation information, see [“Installing Containerized Contrail Clusters Using Server Manager” on page 17](#) and *Installing Containerized Contrail for Single- and Multi-Node Systems Using Server Manager Lite*

Related Documentation

- [Introduction to Containerized Contrail Modules on page 11](#)
- [Contrail Roles Overview](#)
- [Installing Containerized Contrail Clusters Using Server Manager on page 17](#)
- [Installing Containerized Contrail Using Server Manager Lite \(SM-Lite\) on page 20](#)
- [Upgrading Contrail 3.2 to 4.0 on page 285](#)
- [Download Software](#)

Installing Containerized Contrail Clusters Using Server Manager

This topic presents the steps needed to install containerized Contrail Release 4.0 in a single- or multi-node configuration.

You can use Contrail Server Manager or Server Manager Lite (SM-Lite) to provision containerized Contrail.

This is the procedure for using Server Manager. SM-Lite is typically used for Contrail networking, only.

The installation is completed using the following major activities:

- [Installing Server Manager on page 17](#)
- [Creating Objects with Server Manager and JSONs on page 18](#)
- [Preparing the Target System for Provisioning on page 19](#)
- [Provisioning the System on page 19](#)

Installing Server Manager

Before installing Contrail Release 4.0, you must install Contrail Server Manager on a server running Ubuntu.

1. Install the Server Manager wrapper package:

```
dpkg -i contrail-server-manager-installer_[version~sku].deb
```

2. Install Server Manager and its dependent packages, including docker-engine and Cobbler:

```
cd /opt/contrail/contrail_server_manager/; ./setup.sh --all --hostip=[IP address of SM]
```



NOTE: The `setup.sh` script could fail to start the Docker registry if you are installing over an existing version of Server Manager.

If you encounter the Docker registry failure to start error, use the following workaround:

- a. In the `setup.sh` script, comment out the line containing the `docker run` command.
- b. `dpkg --purge contrail-server-manager`
- c. `setup.sh --all --hostip=[IP address of SM]`
3. When the Server Manager install completes with no errors, modify the DHCP template at `/etc/cobbler/dhcp.template` to include the details of the subnet being reimaged or provisioned. Be sure to include DNS details.



NOTE: Container hosts require Internet connectivity at this point to launch the containers.

4. Start the Server Manager process:

```
service contrail-server-manager start
```

For more details about the Server Manager installation process, refer to [“Installing Server Manager” on page 173](#).

Creating Objects with Server Manager and JSONs

Once Server Manager is installed, use Server Manager commands with a JSON file to create Contrail objects.

Configure an appropriate JSON file with the IP addresses, interface names, and password strings specific to your system.

Select a sample JSON from the following and update it to match your system:

- Sample JSONs for an All-In-One-Node Cluster:
[Sample JSONs for an all-in-one, single node with roles](#)
- Sample JSONs for a Multinode Cluster with Two Nodes:
[Sample JSONs for a Multinode Cluster](#)
- Sample JSONs for a Multinode Cluster with 7 Nodes and High Availability
[Sample JSONs for a Multinode Cluster with High Availability:](#)

The following procedure helps you create a target system that includes the components for OpenStack, Contrail controller, analytics, analytics database, and agent. The controller, analytics, and analytics database services are provisioned using Contrail containers, however, the agent service is configured on the bare-metal target host.

1. Configure the images needed for reimagining and provisioning.

- a. Add the Ubuntu image from JSON (used for reimagining)

```
server-manager add image -f image-ubuntu-14.04.04.json
```

- b. Add the Contrail Debian image and containers from JSON (used for provisioning)

```
server-manager add image -f contrail_image.json
```



NOTE: Wait for this command to complete, it operates in the background and can take as long as 5 minutes to complete.

Before proceeding, check for a log message: **Image add/Modify success**, in `/var/log/contrail-server-manager/debug.log`.

2. Configure the cluster(s).

For an all-in-one, single-node demo system:

```
server-manager add cluster -f <all_ins_one_cluster>.json
```

For a multi-node system:

```
server-manager add cluster -f <multi_node_cluster>.json
```

If a Keystone admin password is generated, be sure to write it down.



NOTE: During installation, if a password is provided, no other passwords are generated. If a password is *NOT* provided, all needed passwords are generated.

3. Configure the server.

```
server-manager add server -f contrail_server.json
```

Repeat this step for every server in the system, using the correct **server.json** file, based on the number of servers or type of your system.

Preparing the Target System for Provisioning

To prepare the target system for provisioning, reimage the target system(s), including the Contrail server and the OpenStack server.

- For an all-in-one, single-node demo system:

```
server-manager reimage --server_id <server_id> <ubuntu_image>
```

- For a multi-node system:

```
server-manager reimage --cluster_id <multi_node> <ubuntu_image>
```

Provisioning the System

Launch the system provisioning.

- For an all-in-one, single-node demo system:

```
server-manager provision --cluster_id <all_in_one_cluster> combined_image_mainline
```

- For a multi-node system:

```
server-manager provision --cluster_id <multi_node> combined_image_mainline
```

The **server-manager provision** command first provisions the OpenStack role, which includes using Puppet manifests. Next, the command provisions Contrail Docker containers and compute nodes.

You can monitor progress of the provisioning by observing log entries:

```
/var/log/contrail-server-manager/debug.log
```

When provisioning is complete, confirm successful installation by creating a virtual network and launching virtual machines from the OpenStack node.

**Related
Documentation**

- Sample JSONs for an All-In-One-Node Cluster:
[Sample JSONs for an all-in-one, single node with roles](#)
- Sample JSONs for a Multinode Cluster with Two Nodes:
[Sample JSONs for a Multinode Cluster](#)
- Sample JSONs for a Multinode Cluster with 7 Nodes and High Availability
[Sample JSONs for a Multinode Cluster with High Availability](#)
- [Introduction to Containerized Contrail Modules on page 11](#)
- [Contrail Roles Overview](#)
- [Installing the Operating System and Contrail Packages on page 15](#)
- [Installing Containerized Contrail Using Server Manager Lite \(SM-Lite\) on page 20](#)
- [Upgrading Contrail 3.2 to 4.0 on page 285](#)

Installing Containerized Contrail Using Server Manager Lite (SM-Lite)

Server Manager Lite (SM-Lite) is a streamlined version of Server Manager. SM-Lite has functionality similar to Server Manager, except it does not perform reimaging. SM-Lite is typically used for Contrail networking only.

You can use Contrail Server Manager or Server Manager Lite (SM-Lite) to provision containerized Contrail. To use SM-Lite for provisioning, you install regular Server Manager, then use SM-Lite commands for provisioning.

This topic is the procedure for installing and provisioning Contrail 4.0 and later using SM-Lite.

The SM-Lite installation of containerized Contrail is completed using the following major activities:

- [Preparing for SM-Lite Installation on page 20](#)
- [Installing SM-Lite on page 21](#)
- [Provisioning Contrail Using SM-Lite on page 22](#)
- [Sample JSONs and Testbed.py on page 22](#)

Preparing for SM-Lite Installation

For Contrail 4.0, SM-Lite install is only supported on Ubuntu 14.04.5. Contrail 4.1 adds support for Ubuntu 16.04.2.

Before installing containerized Contrail, you must install Server Manager SM-Lite on a server running a supported version of Ubuntu.

You can install SM-Lite on any server or node, and you can run it using multiple options:

- Use a single node or VM to provision Contrail, then install SM-Lite on the same node and perform Contrail provisioning.
- Use a separate node or VM to install SM-Lite, and provision Contrail with the rest of the nodes.
- Use a node or VM that has Contrail roles (typically a config node) to install SM-Lite.

To specify servers and associated Contrail roles and cluster details, you can use either a **testbed.py** or JSONs based on the sample JSONs used with regular Server Manager. The image details come from the image JSON.

Prerequisites

Before installing the SM-Lite package, ensure the following cautions have been met:

- Ensure that the **sources.list** is present and empty.
- Ensure that **/etc/apt/sources.list.d/** is not pointing to any external or local repositories.

If you are installing SM-Lite on a VM spawned from OpenStack Horizon or from an Ubuntu cloud image:

- Verify that the VM is set up correctly with hostname and domain details:
 - The hostname and domain name are present in **/etc/hosts** as follows:


```
<Host non mgmt IP> <server hostname>.<domain_name> <server hostname>
```
 - The domain name is present in **/etc/resolv.conf** as follows:


```
search <domain_name>
```
- When correctly set up, the command "**hostname -f**" will return **< hostname >.< domain_name >**

Installing SM-Lite

1. Install the regular Server Manager wrapper package (Debian). (An example package is: **contrail-server-manager-installer_2.22~juno_all.deb**.)

```
dpkg -i </github-build/mainline/<build_number> /ubuntu-14-04/mitaka/artifacts/
contrail-server-manager-installer_4.0.0.0-<build-number>~mitaka_all.deb>
```

2. Now you can use the SM-Lite **provision_containers** command to provision Contrail.

The full syntax and available options of the **provision_containers.sh** script:

```
Help:
`/opt/contrail/contrail_server_manager/provision_containers.sh -h`
`-h --help`
`-cj <cluster json path>`
`-sj <server json path>`
`-ij <image json path>`
`-t|--testbed <testbed.py path>`
`-c <contrail cloud docker package path>`
```

```
`-cid|--cluster-id <cluster-id>`  
`-ni|--no-install-sm-lite`
```

The **-ni** option is used to reprovision an existing cluster, create a new cluster, or upgrade an existing cluster with a different version.

For more details about SM-Lite, refer to [“Installing and Using Server Manager Lite” on page 225](#).

Provisioning Contrail Using SM-Lite

To activate SM-Lite and provision the target systems, use **provision_containers.sh** along with system-specific configuration information.

Provision Contrail with system-specific configuration information using one of the following options:

- Using JSONs

```
/opt/contrail/contrail_server_manager/provision_containers.sh -cj <cluster json path>  
-sj <server json path> -ij <image json path> --cluster-id <Cluster ID>
```

- Using **testbed.py** and **contrail-docker-cloud.tgz**

```
/opt/contrail/contrail_server_manager/provision_containers.sh -t <testbed.py path> -c  
<contrail-cloud-docker tgz path> --cluster-id <Cluster ID>
```

The SM-Lite provisioning logs can be viewed at
/var/log/contrail-server-manager/debug.log.

Running the **provision_containers.sh** script does the following:

1. Installs SM-Lite components: sm client, sm webui, sm monitoring/inventory, and the like.
2. Prepares the targets for provisioning by running the **preconfig.py** script.
3. Adds Server Manager objects for cluster, server, and image from the JSONs or the **testbed.py** as provided.
4. Loads Docker containers and pushes them to the registry in the background.
5. Launches the Contrail provisioning, using the Server Manager client CLI.

Sample JSONs and Testbed.py

Use the SM-Lite command **provision_containers.sh** with a JSON file or a **testbed.py** to provision Contrail objects.

Configure an appropriate JSON file or **testbed.py** with the IP addresses, interface names, and password strings specific to your system, then identify its path when you use the SM-Lite **provision_containers.sh** command.

Select a sample JSON or **testbed.py** from the following and update it to match your system:

- [Sample testbed.py for Provisioning Containers with SM-Lite](#)
- [Sample JSONs for an All-In-One-Node Cluster \(for demo\)](#)
- [Sample JSONs for a Multinode Cluster with Two Nodes](#)
- [Sample JSONs for a Multinode Cluster with 7 Nodes and High Availability](#)

Related Documentation

- [Sample JSONs for an All-In-One-Node Cluster \(for demo\)](#)
- [Sample JSONs for a Multinode Cluster with Two Nodes](#)
- [Sample JSONs for a Multinode Cluster with 7 Nodes and High Availability](#)
- [Sample testbed.py for Provisioning Containers with SM-Lite](#)
- [Introduction to Containerized Contrail Modules on page 11](#)
- [Contrail Roles Overview](#)
- [Installing the Operating System and Contrail Packages on page 15](#)
- [Installing Containerized Contrail Clusters Using Server Manager on page 17](#)
- [Upgrading Contrail 3.2 to 4.0 on page 285](#)

Supporting Multiple Interfaces on Servers and Nodes

This section describes how to set up and manage multiple interfaces.

- [Support for Multiple Interfaces on page 23](#)
- [Server Interface Examples on page 25](#)
- [Interface Naming and Configuration Management on page 25](#)

Support for Multiple Interfaces

Servers and nodes with multiple interfaces should be deployed with exclusive management and control and data networks. In the case of multiple interfaces per server, the expectation is that the management network provides only management connectivity to the cluster, and the control and data network carries the control plane information and the guest traffic data.

Examples of control traffic include the following:

- XMPP traffic between the control nodes and the compute nodes.
- BGP protocol messages across the control nodes.
- Statistics, monitoring, and health check data collected by the analytics engine from different parts of the system.

In Contrail, control and data must share the same interface, configured in the **testbed.py** file in a section named **control_data**.

Number of cfgm Nodes Supported

The Contrail system can have any number of **cfgm** nodes.

Uneven Number of Database Nodes Required

In Contrail, Apache ZooKeeper resides on the database node. Because a ZooKeeper ensemble operates most effectively with an odd number of nodes, it is required to have an odd number (3, 5, 7, and so on) of database nodes in a Contrail system.

Support for VLAN Interfaces

A VLAN ID can also be specified in the **server.json** file under the **network, interfaces** section, similar to the following example:

```
    "network": {
      "interfaces": [
        {
          "name": "vlan2003",
          "type": "vlan",
          "vlan": "2003",
          "parent_interface": "bond0",
          "ip_address": "10.224.11.10/24",
          "default_gateway": "10.224.12.1"
        }
      ]
    }
```

Support for Bonding Options

Contrail provides support for bond interface options.

The default bond interface options are:

miimon=100, mode=802.3ad(lacp), xmit_hash_policy=layer3+4

For Contrail 4.0 and later, in the **server.json** bond section, anything other than name and member are treated as a bond interface option, and provisioned as such. The following is an example:

```
    "network": {
      "interfaces": [
        {
          "name": "bond0",
          "type": "bond",
          "bond_options": { "miimon": "100", "mode": "802.3ad",
            "xmit_hash_policy": "layer3+4" },
          "member_interfaces": ["p20p1", "p20p2"]
        }
      ],
    }
```

Support for Static Route Options

Contrail provides support for adding static routes on target systems. This option is ideal for use cases in which a system has servers with multiple interfaces and has control data or management connections that span multiple networks.

The following shows static routes added in the **server.json** under the 'network' section.


```

"network": {
  "routes": [
    {
      "gateway": "3.3.2.254",
      "interface": "enp129s0f0",
      "netmask": "255.255.255.0",
      "network": "3.3.4.0"
    },
    {
      "gateway": "3.3.3.254",
      "interface": "enp129s0f1",
      "netmask": "255.255.255.0",
      "network": "3.3.5.0"
    }
  ]
}

```

Server Interface Examples

In Contrail Release 1.10 and later, control and data are required to share the same interface. A set of servers can be deployed in any of the following combinations for management, control, and data:

- **mgmt=control=data** -- Single interface use case
- **mgmt, control=data** -- Exclusive management access, with control and data sharing a single network.

In Contrail, the following server interface combinations are not allowed:

- **mgmt=control, data--** Dual interfaces in Layer 3 mode, management and control shared on a single network
- **mgmt, control, data**—Complete exclusivity across management, control, and data traffic.

Interface Naming and Configuration Management

On a standard Linux installation there is no guarantee that a physical interface will come up with the same name after a system reboot. Linux NetworkManager tries to accommodate this behavior by linking the interface configurations to the hardware addresses of the physical ports. However, Contrail avoids using hardware-based configuration files because this type of solution cannot scale when using remote provisioning and management techniques.

The Contrail alternative is a threefold interface-naming scheme based on **<bus, device, port (or function)>**. As an example, on a server operating system that typically assigns interface names such as **p4p0** and **p4p1** for onboard interfaces, the Contrail system assigns **p4p0p0** and **p4p0p1**, when using the optional **contrail-interface-name** package.

When the **contrail-interface-name** package is installed, it uses the threefold naming scheme to provide consistent interface naming after reboots. The **contrail-interface-name** package is installed by default when a Contrail ISO image is installed. If you are using an

RPM-based installation, you should install the **contrail-interface-name** package before doing any network configuration.

If your system already has another mechanism for getting consistent interface names after a reboot, it is not necessary to install the **contrail-interface-name** package.

Related Documentation

- *Juniper OpenStack High Availability*

Configuring the Control Node

An important task after a successful installation is to configure the control node. This procedure shows how to configure basic BGP peering between one or more virtual network controller control nodes and any external BGP speakers. External BGP speakers, such as Juniper Networks MX80 routers, are needed for connectivity to instances on the virtual network from an external infrastructure or a public network.

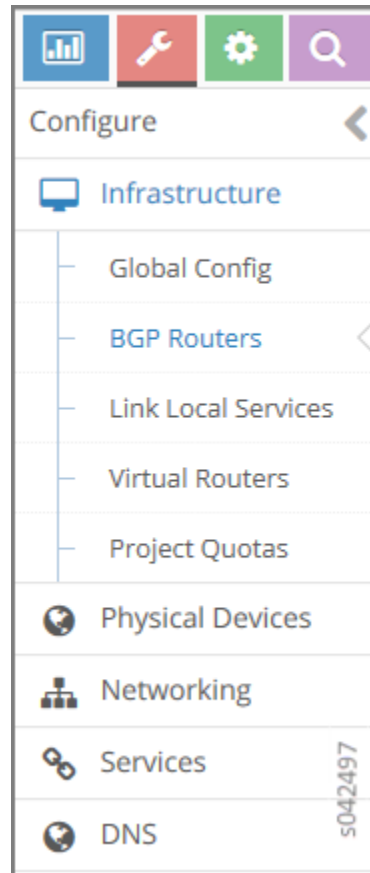
Before you begin, ensure that the following tasks are completed:

- The Contrail Controller base system image has been installed on all servers.
- The role-based services have been assigned and provisioned.
- IP connectivity has been verified between all nodes of the Contrail Controller.
- You can access the Contrail user interface at **<http://nn.nn.nn.nn:8080>**, where ***nn.nn.nn.nn*** is the IP address of the configuration node server that is running the **contrail-webui** service.

To configure BGP peering in the control node:

1. From the Contrail Controller module control node (<http://nn.nn.nn.nn:8080>), select **Configure > Infrastructure > BGP Routers**; see [Figure 2 on page 28](#).

Figure 2: Configure > Infrastructure > BGP Routers



A summary screen of the control nodes and BGP routers is displayed; see [Figure 3 on page 28](#).

Figure 3: BGP Routers Summary

Configure > Infrastructure > BGP Routers				
BGP Routers				
<input type="checkbox"/> IP Address	Type	Vendor	HostName	
▶ <input type="checkbox"/> 10.84.25.31	Control Node	contrail	b5s31	⚙
▶ <input type="checkbox"/> 10.84.11.252	BGP Router	mx	a3-mx80-1	⚙
▶ <input type="checkbox"/> 10.84.25.30	Control Node	contrail	b5s30	⚙
▶ <input type="checkbox"/> 10.84.25.29	Control Node	contrail	b5s29	⚙
▶ <input type="checkbox"/> 10.84.25.28	Control Node	contrail	b5s28	⚙
▶ <input type="checkbox"/> 10.84.25.27	Control Node	contrail	b5s27	⚙
▶ <input type="checkbox"/> 10.84.11.253	BGP Router	mx	mx1	⚙
Total: 7 records 50 Records ▼				
Page 1 of 1				

2. (Optional) The global AS number is 64512 by default. To change the AS number, on the **BGP Router** summary screen click the gear wheel and select **Edit**. In the Edit BGP

Router window enter the new number.

3. To create control nodes and BGP routers, on the **BGP Routers** summary screen, click the **+** icon. The **Create BGP Router** window is displayed; see [Figure 4 on page 29](#).

Figure 4: Create BGP Router

4. In the **Create BGP Router** window, click **BGP Router** to add a new BGP router or click **Control Node** to add control nodes.

For each node you want to add, populate the fields with values for your system. See [Table 4 on page 29](#).

Table 4: Create BGP Router Fields

Field	Description
Hostname	Enter a name for the node being added.
Vendor ID	Required for external peers. Populate with a text identifier, for example, "MX-0". (BGP peer only)
IP Address	The IP address of the node.
Router ID	Enter the router ID.
Autonomous System	Enter the AS number for the node. (BGP peer only)

Table 4: Create BGP Router Fields (*continued*)

Field	Description
Address Families	Enter the address family, for example, inet-vpn
Hold Time	BGP session hold time. The default is 90 seconds; change if needed.
BGP Port	The default is 179; change if needed.
Authentication Mode	Enable MD5 authentication if desired.
Authentication key	Enter the Authentication Key value.
Physical Router	The type of the physical router.
Available Peers	Displays peers currently available.
Configured Peers	Displays peers currently configured.

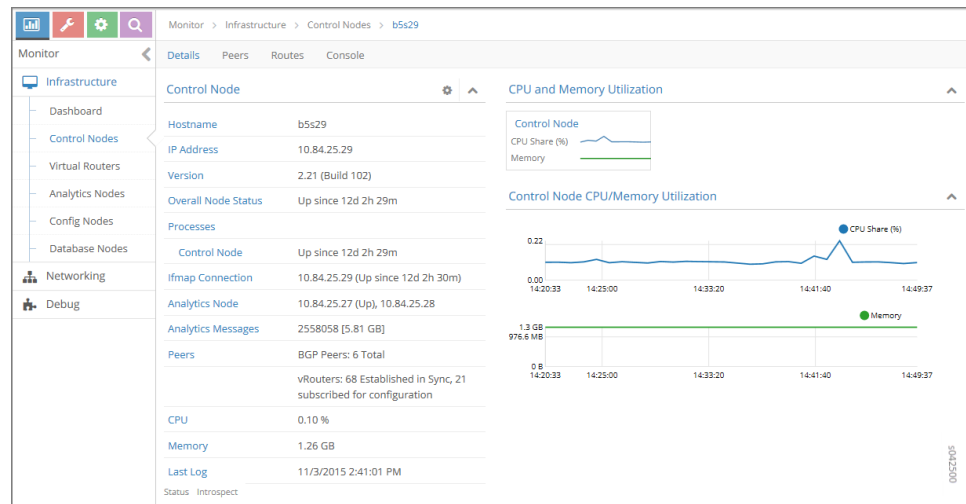
- Click **Save** to add each node that you create.
- To configure an existing node as a peer, select it from the list in the **Available Peers** box, then click >> to move it into the **Configured Peers** box.
Click << to remove a node from the **Configured Peers** box.
- You can check for peers by selecting **Monitor > Infrastructure > Control Nodes**; see [Figure 5 on page 30](#).

Figure 5: Control Nodes



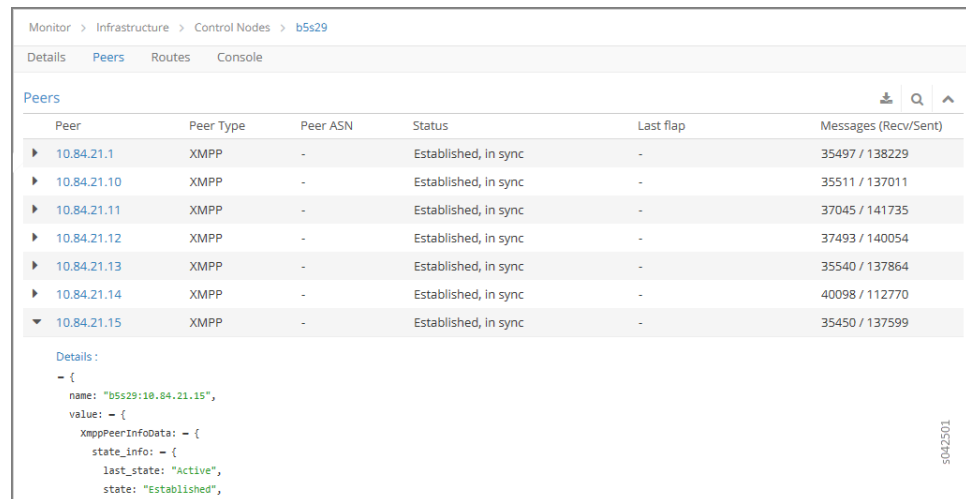
In the **Control Nodes** window, click any hostname in the memory map to view its details; see [Figure 6 on page 31](#).

Figure 6: Control Node Details



8. Click the **Peers** tab to view the peers of a control node; see [Figure 7 on page 31](#).

Figure 7: Control Node Peers Tab



Related Documentation

- [Creating a Virtual Network with Juniper Networks Contrail on page 296](#)
- [Creating a Virtual Network with OpenStack Contrail on page 299](#)

Adding a New Node to an Existing Containerized Contrail Cluster

This is the initial process for adding a new node to an existing cluster in containerized Contrail.

- [Controller Configuration on page 32](#)

Controller Configuration

1. Create contrailctl configuration and start a controller container on a new node.

- Configure contrailctl configurations in `/etc/contrailctl/controller.conf`.

See examples on github at:

[contrail-docker/tools/python-contrailctl/examples/configs/controller.conf](https://github.com/contrail-io/contrail-docker/tree/master/tools/python-contrailctl/examples/configs/controller.conf)

- Start the controller container. For more information, see [How to run Contrail Docker containers](#).
 - Wait for the new containers to come up completely.
2. Configure the existing cluster nodes with new nodes.

The purpose of this step is to reconfigure the existing cluster application configurations to include newly added servers, then restart to accommodate the configuration changes.

You can do this by using one of two methods described below:

- [Using contrailctl to add node configuration on existing containers on page 32](#)
- [Manually configure contrailctl on all containers and sync the configs on page 33](#)

Using contrailctl to add node configuration on existing containers

You can use contrailctl to add the node configuration on existing containers by running the following steps on all existing containers on all cluster nodes.



NOTE: Run this step first on all zookeeper *follower* nodes, then run on the leader node.

1. Determine which node is the leader node.

To determine which node is the leader and which are followers in a zookeeper cluster, run the following commands against your zookeeper cluster nodes.

```
$ echo stat | nc 192.168.0.102 2181 | grep Mode Mode: leader
```

```
$ echo stat | nc 192.168.0.100 2181 | grep Mode Mode: follower
```

2. Run contrailctl on all the existing containers in all cluster nodes, follower nodes first and leader node last.

```
$ contrailctl config node add -h
usage: contrailctl config node add [-h] -t {controller,analyticsdb,analytics}
                                     -n NODE_ADDRESSES [-s SEED_LIST]
                                     [-f CONFIG_FILE] -c
                                     {controller,analyticsdb,analytics,agent,lb,kubemanager,mesosmanager}
                                     [--config-list CONFIG_LIST]
```



```

optional arguments:
  -h, --help                show this help message and exit
  -t {controller,analyticsdb,analytics}, --type
{controller,analyticsdb,analytics}
                        Type of node
  -n NODE_ADDRESSES, --node-addresses NODE_ADDRESSES
                        Comma separated list of node addresses
  -s SEED_LIST, --seed-list SEED_LIST
                        Comma separated list of seed nodes to be used
  -f CONFIG_FILE, --config-file CONFIG_FILE
                        Master config file path
  -c {controller,analyticsdb,analytics,agent,lb,kubemanager,mesosmanager},
--component
{controller,analyticsdb,analytics,agent,lb,kubemanager,mesosmanager}
                        contrail role to be configured
  --config-list CONFIG_LIST
                        comma separated list of config nodes. Optional it
is
                        needed only when the new controller nodes added are
                        config service disabled

# Add new controllers in analytics container
$ contrailctl config node add -t controller -n 192.168.0.10,192.168.0.11
-s 192.168.0.102,192.168.0.99 -c analytics

# Add new controllers in analyticsdb container
$ contrailctl config node add -t controller -n 192.168.0.10,192.168.0.11
-s 192.168.0.102,192.168.0.99 -c analyticsdb

# Add new controllers in other controller containers
$ contrailctl config node add -t controller -n 192.168.0.10,192.168.0.11
-s 192.168.0.102,192.168.0.99 -c controller

```

Manually configure contrailctl on all containers and sync the configs

1. Determine which node is the leader node.

To determine which node is the leader and which are followers in a zookeeper cluster, run the following commands against your zookeeper cluster nodes.

```
$ echo stat | nc 192.168.0.102 2181 | grep Mode Mode: leader
```

```
$ echo stat | nc 192.168.0.100 2181 | grep Mode Mode: follower
```

2. Manually configure `/etc/contrailctl/controller.conf` with new nodes for various `*_list` configurations and `config_seed_list`. See examples at:

<https://github.com/Juniper/contrail-docker/blob/master/tools/python-contrailctl/examples/configs/controller.conf>

3. Run `contrailctl` within the containers.

```
$ docker exec <container name> contrailctl config sync -c <component name>
```

```
$ docker exec controller contrailctl config sync -c controller
```

Removing Nodes in an Existing Containerized Cluster

For the first version of containerized Contrail, there is no script available for removing a node from an existing cluster. If it is necessary to remove a node from an existing containerized Contrail cluster, please contact Juniper Networks JTAC for assistance.

Using contrailctl to Configure Services Within Containers

Starting with Contrail 4.0, some subsystems of Contrail are delivered as Docker containers. The **contrailctl** tool is a set of commands that enable a user to make some changes to the configuration file within a Contrail container.

- [What is contrailctl? on page 34](#)
- [Command Operations on page 34](#)

What is contrailctl?

Starting with Contrail 4.0, some modules of the Contrail architecture have been grouped by function into Docker containers. Each container has an INI-based configuration file to maintain the specific configuration for that container. The **contrailctl** is a tool within the container that provides the user a simple command structure for provisioning and operating the Contrail services packaged in the container.

Because it is complex to provision and manage the various services within Contrail containers, the **contrailctl** tool helps configure the services in the container to be in sync with the container-specific configuration files.

The **contrailctl** tool is driven by the single INI-based configuration file per container, for example, the **/etc/contrailctl/controller.conf** for the controller container. Any state changes of the services within the container must be made according to the configuration in the **contrailctl** configuration file for that container. The **contrailctl** configuration files are available on each node at a default location of **/etc/contrailctl/*.conf**.

Any changes made to the configuration files in the node are available within the container.

Each Contrail container has a separate **contrailctl** configuration file, currently:

- **contrail-controller**—**/etc/contrailctl/controller.conf**
- **contrail-analytics**—**/etc/contrailctl/analytics.conf**
- **contrail-analyticsdb**—**/etc/contrailctl/analyticsdb.conf**

Sample container configuration files can be seen at

<https://github.com/Juniper/contrail-docker/tree/master/tools/python-contrailctl/examples/configs>

Command Operations

The **contrailctl** is used within the node that holds a container. It is used at startup to configure and start the services within the container. The user must connect to the container to run **contrailctl**, or use the following command syntax to run **contrailctl**:

```
docker exec <container name or id> contrailctl <arguments>
```

Example:

```
docker exec controller contrailctl config sync -c controller -Fv
```

The main function of the **contrailctl** is to ensure that the desired configurations for the services within a container are in sync with the **contrailctl** master configuration file within the container.

Command Syntax and Options

```
$ contrailctl config sync -h
usage: contrailctl config sync [-h] [-f CONFIG_FILE] -c
{controller,analyticsdb,analytics,agent,lb,kubemanager,mesosmanager}
[-F] [-t TAGS]

optional arguments:
  -h, --help            show this help message and exit
  -f CONFIG_FILE, --config-file CONFIG_FILE
                        Master config file path
  -c {controller,analyticsdb,analytics,agent,lb,kubemanager,mesosmanager},
  --component {controller,analyticsdb,analytics,agent,lb,kubemanager,mesosmanager}
                        Component[s] to be configured
  -F, --force            Whether to apply config forcibly
  -t TAGS, --tags TAGS  comma separated list of tags to runspecific set of
                        ansible code
```

Updating and Syncing Service Configurations Within the Container

You can update service configurations by editing the appropriate container configuration file and then syncing.

While starting the container, the **contrailctl** configurations are provided under **/etc/contrailctl**. During startup, **contrailctl config sync** runs to synchronize the configurations to the internal services.

If a user wants to add or change configurations, the user can edit the appropriate configuration file in **/etc/contrailctl/** and then manually run **contrailctl config sync** on that specific container.

Using **contrailctl config sync** synchronizes the entire configuration from the master configuration file in **/etc/contrailctl** to the service configurations within the container.

Syntax and Usage: config sync

```
contrailctl config sync [section] [param] [-f|--force]
```

- Use the options **section** and **parameter** to restrict the data to be synced to a specific section and parameter.
- Use the optional **force** to perform an Ansible run, even if there is no configuration change to be synced.

Example: config sync

In this example, the user wants to add a configuration "foo=bar" to the controller container after the container is started.

The following example shows the procedure to sync a configuration change within the controller container.

1. The user edits the `/etc/contrailctl/controller.conf` to add the desired configuration changes within the node that holds the container.
2. The user syncs the change to the services running within the container.

```
$ docker exec <my controller> config sync -c controller -v
```

Related Documentation

- [Introduction to Containerized Contrail Modules on page 11](#)

Supporting Multiple Interfaces on Servers and Nodes

This section describes how to set up and manage multiple interfaces.

- [Support for Multiple Interfaces on page 36](#)
- [Server Interface Examples on page 38](#)
- [Interface Naming and Configuration Management on page 38](#)

Support for Multiple Interfaces

Servers and nodes with multiple interfaces should be deployed with exclusive management and control and data networks. In the case of multiple interfaces per server, the expectation is that the management network provides only management connectivity to the cluster, and the control and data network carries the control plane information and the guest traffic data.

Examples of control traffic include the following:

- XMPP traffic between the control nodes and the compute nodes.
- BGP protocol messages across the control nodes.
- Statistics, monitoring, and health check data collected by the analytics engine from different parts of the system.

In Contrail, control and data must share the same interface, configured in the `testbed.py` file in a section named `control_data`.

Number of cfm Nodes Supported

The Contrail system can have any number of `cfm` nodes.

Uneven Number of Database Nodes Required

In Contrail, Apache ZooKeeper resides on the database node. Because a ZooKeeper ensemble operates most effectively with an odd number of nodes, it is required to have an odd number (3, 5, 7, and so on) of database nodes in a Contrail system.

Support for VLAN Interfaces

A VLAN ID can also be specified in the **server.json** file under the **network, interfaces** section, similar to the following example:

```

"network": {
  "interfaces": [
    {
      "name": "vlan2003",
      "type": "vlan",
      "vlan": "2003",
      "parent_interface": "bond0",
      "ip_address": "10.224.11.10/24",
      "default_gateway": "10.224.12.1"
    }
  ]
}

```

Support for Bonding Options

Contrail provides support for bond interface options.

The default bond interface options are:

miimon=100, mode=802.3ad(lacp), xmit_hash_policy=layer3+4

For Contrail 4.0 and later, in the **server.json** bond section, anything other than name and member are treated as a bond interface option, and provisioned as such. The following is an example:

```

"network": {
  "interfaces": [
    {
      "name": "bond0",
      "type": "bond",
      "bond_options": { "miimon": "100", "mode": "802.3ad",
        "xmit_hash_policy": "layer3+4" },
      "member_interfaces": [ "p20p1", "p20p2" ]
    }
  ],
}

```

Support for Static Route Options

Contrail provides support for adding static routes on target systems. This option is ideal for use cases in which a system has servers with multiple interfaces and has control data or management connections that span multiple networks.

The following shows static routes added in the **server.json** under the 'network' section.

```

"network": {
  "routes": [
    {
      "gateway": "3.3.2.254",

```

```
        "interface": "enp129s0f0",
        "netmask": "255.255.255.0",
        "network": "3.3.4.0"
    },
    {
        "gateway": "3.3.3.254",
        "interface": "enp129s0f1",
        "netmask": "255.255.255.0",
        "network": "3.3.5.0"
    }
  ]
}
```

Server Interface Examples

In Contrail Release 1.10 and later, control and data are required to share the same interface. A set of servers can be deployed in any of the following combinations for management, control, and data:

- **mgmt=control=data** -- Single interface use case
- **mgmt, control=data** -- Exclusive management access, with control and data sharing a single network.

In Contrail, the following server interface combinations are not allowed:

- **mgmt=control, data**--Dual interfaces in Layer 3 mode, management and control shared on a single network
- **mgmt, control, data**—Complete exclusivity across management, control, and data traffic.

Interface Naming and Configuration Management

On a standard Linux installation there is no guarantee that a physical interface will come up with the same name after a system reboot. Linux NetworkManager tries to accommodate this behavior by linking the interface configurations to the hardware addresses of the physical ports. However, Contrail avoids using hardware-based configuration files because this type of solution cannot scale when using remote provisioning and management techniques.

The Contrail alternative is a threefold interface-naming scheme based on **<bus, device, port (or function)>**. As an example, on a server operating system that typically assigns interface names such as **p4p0** and **p4p1** for onboard interfaces, the Contrail system assigns **p4p0p0** and **p4p0p1**, when using the optional **contrail-interface-name** package.

When the **contrail-interface-name** package is installed, it uses the threefold naming scheme to provide consistent interface naming after reboots. The **contrail-interface-name** package is installed by default when a Contrail ISO image is installed. If you are using an RPM-based installation, you should install the **contrail-interface-name** package before doing any network configuration.

If your system already has another mechanism for getting consistent interface names after a reboot, it is not necessary to install the **contrail-interface-name** package.

Related Documentation

- *Juniper OpenStack High Availability*

Contrail Global Controller

Starting with Release 3.1, Contrail provides support for a global controller. The global controller feature provides a seamless controller experience across multiple regions in a cloud environment by helping manage multiple OpenStack installations, each having its own Keystone, Neutron, Nova and so on. High availability is provided by using separate failure domains by region.

To handle the resource burdens when connecting and configuring servers and virtual machines over multiple, different regions, the global controller has the following main responsibilities:

- [Resource Identifier Management on page 39](#)
- [Multiple Location Resource Provisioning on page 39](#)

Resource Identifier Management

The global controller uses centralized resource ID management to manage multiple types of identifiers (IDs), identifying such things as route targets, virtual networks, security groups, and so on.

The Contrail global controller can interconnect virtual networks (VNs) residing in different data centers using BGP VPN technology. BGP VPN recognizes virtual private networks (VPNs) by using route target identifiers. A virtual network ID is used to identify the same virtual networks in different data centers, to prevent looping in service chains. Security group IDs identify the same security group over multiple data centers, so that the same security group policies can be used. It is important to use the same security group over multiple regions to allow traffic from all routes in the same virtual networks.

The global controller needs to manage all of the identifiers when interconnecting multiple data centers.

Multiple Location Resource Provisioning

There are many cases in which the same resource, such as policy or services, needs to exist in multiple data centers. For example, there might be a security policy to apply a firewall for any traffic for an application server network that exists in multiple locations. Each location needs to have the same virtual network, network policy, and firewalls. The Contrail global controller automates this process.

Requirements, Assumptions, and Constraints

The following are requirements, assumptions, and constraints for implementing the Contrail global controller:

- Each data center has different regions with OpenStack with Contrail.
- Each region that is managed under the same OpenStack Keystone or Keystone data must be replicated with multiple data centers.
- The global controller has a secure API connection for each OpenStack with Contrail region.
- Each Contrail controller needs peering by eBGP or iBGP; eBGP is recommended.
- Each OpenStack Keystone has an administrator account for the global controller. The account must be authorized to manage resources in each region.

Platform Support

The following are the platform requirements for the Contrail global controller:

- OpenStack Liberty
- Ubuntu 14.04.4
- Contrail Release 3.1 or greater

Installation

The global controller is a new feature starting with Contrail Release 3.1. The installation instructions can be found in the following location:

<https://nati.gitbooks.io/contrail-global-controller/content/doc/installation.html>

Related Documentation

- *Contrail Global Controller Web User Interface*

Role- and Resource-Based Access Control

- [Contrail Role- and Resource-Based Access \(RBAC\) Overview on page 41](#)
- [API-Level Access Control on page 41](#)
- [Object Level Access Control on page 42](#)
- [Configuration on page 42](#)
- [Utilities on page 44](#)
- [Upgrading from Previous Releases on page 45](#)
- [Configuring RBAC Using the Contrail User Interface on page 46](#)
- [RBAC Resources on page 48](#)

Contrail Role- and Resource-Based Access (RBAC) Overview

Contrail Release 3.0 and later provides role- and resource-based access control (RBAC) with API operation-level access control.

The RBAC implementation relies on user credentials obtained from Keystone from a token present in an API request. Credentials include user, role, tenant, and domain information.

API-level access is controlled by a list of rules. The attachment points for the rules include **global-system-config**, domain, and project. Resource-level access is controlled by permissions embedded in the object.

API-Level Access Control

If the RBAC feature is enabled, the API server requires a valid token to be present in the **X-Auth-Token** of any incoming request. The API server trades the token for user credentials (role, domain, project, and so on) from Keystone.

If a token is missing or is invalid, an HTTP error 401 is returned.

The **api-access-list** object holds access rules of the following form:

<object, field> => list of <role:CRUD>

Where:

object—An API resource such as network or subnet.

field—Any property or reference within the resource. The **field** option can be multilevel, for example, **network.ipam.host-routes** can be used to identify multiple levels. The **field** is optional, so in its absence, the create, read, update, and delete (CRUD) operation refers to the entire resource.

role—The Keystone role name.

Each rule also specifies the list of roles and their corresponding permissions as a subset of the CRUD operations.

Example: ACL RBAC Object

The following is an example access control list (ACL) object for a project in which the admin and any users with the **Development** role can perform CRUD operations on the network in a project. However, only the **admin** role can perform CRUD operations for policy and IP address management (IPAM) inside a network.

```
<virtual-network, network-policy> => admin:CRUD
<virtual-network, network-ipam> => admin:CRUD
<virtual-network, *> => admin:CRUD, Development:CRUD
```

Rule Sets and ACL Objects

The following are the features of rule sets for access control objects in Contrail.

- The rule set for validation is the union of rules from the ACL attached to:
 - User project
 - User domain
 - Default domainIt is possible for the project or domain access object to be empty.
- Access is only granted if a rule in the combined rule set allows access.
- There is no explicit deny rule.
- An ACL object can be shared within a domain. Therefore, multiple projects can point to the same ACL object. You can make an ACL object the default.

Object Level Access Control

The **perms2** permission property of an object allows fine-grained access control per resource.

The **perms2** property has the following fields:

owner —This field is populated at the time of creation with the tenant UUID value extracted from the token.

share list —The share list gets built when the object is selected for sharing with other users. It is a list of tuples with which the object is shared.

The **permission** field has the following options:

- **R**—Read object
- **W**—Create or update object
- **X**—Link (refer to) object

Access is allowed as follows:

- If the user is the owner and permissions allow (rwx)
- Or if the user tenant is in a shared list and permissions allow
- Or if world access is allowed

Configuration

This section describes the parameters used in Contrail RBAC.

- [Parameter: aaa-mode on page 43](#)
- [Parameter: cloud_admin_role on page 43](#)

- [Global Read-Only Role on page 44](#)
- [Parameter Changes in /etc/neutron/api-paste.ini on page 44](#)

Parameter: `aaa-mode`

RBAC is controlled by a parameter named **aaa-mode**. This parameter is used in place of the multi-tenancy parameter of previous releases.

The **aaa-mode** can be set to the following values:

- **no-auth**—No authentication is performed and full access is granted to all.
- **cloud-admin**—Authentication is performed and only the admin role has access.
- **rbac**—Authentication is performed and access is granted based on role.



NOTE: The `multi_tenancy` parameter is deprecated, starting with Contrail 3.0. The parameter should be removed from the configuration. Instead, use the `aaa_mode` parameter for RBAC to take effect.

If the `multi_tenancy` parameter is not removed, the `aaa-mode` setting is ignored.

Parameter: `cloud_admin_role`

A user who is assigned the **cloud_admin_role** has full access to everything.

This role name is configured with the **cloud_admin_role** parameter in the API server. The default setting for the parameter is **admin**. This role must be configured in Keystone to change the default value.

If a user has the **cloud_admin_role** in one tenant, and the user has a role in other tenants, then the **cloud_admin_role** role must be included in the other tenants. A user with the **cloud_admin_role** doesn't need to have a role in all tenants, however, if that user has any role in another tenant, that tenant must include the **cloud_admin_role**.

Configuration Files with Cloud Admin Credentials

The following configuration files contain **cloud_admin_role** credentials:

- `/etc/contrail/contrail-keystone-auth.conf`
- `/etc/neutron/plugins/opencontrail/ContrailPlugin.ini`
- `/etc/contrail/contrail-webui-userauth.js`

Changing Cloud Admin Configuration Files

Modify the cloud admin credential files if the **cloud_admin_role** role is changed.

1. Change the configuration files with the new information.
2. Restart the following:

- API server
`service supervisor-config restart`
- Neutron server
`service neutron-server restart`
- WebUI
`service supervisor-webui restart`

Global Read-Only Role

You can configure a global read-only role (**global_read_only_role**).

A **global_read_only_role** allows read-only access to all Contrail resources. The **global_read_only_role** must be configured in Keystone. The default **global_read_only_role** is not set to any value.

A **global_read_only_role** user can use the Contrail Web UI to view the global configuration of Contrail default settings.

Setting the Global Read-Only Role

To set the global read-only role:

1. The **cloud_admin** user sets the **global_read_only_role** in the Contrail API:

```
/etc/contrail/contrail-api.conf  
  
global_read_only_role = <new-admin-read-role>
```

2. Restart the **contrail-api** service:

```
service contrail-api restart
```

Parameter Changes in /etc/neutron/api-paste.ini

Contrail RBAC operation is based upon a user token received in the **X-Auth-Token** header in API requests. The following change must be made in **/etc/neutron/api-paste.ini** to force Neutron to pass the user token in requests to the Contrail API server:

```
keystone = user_token request_id catch_errors ....  
...  
...  
[filter:user_token]  
paste.filter_factory =  
neutron_plugin_contrail.plugins.opencontrail.neutron_middleware:token_factory
```

Utilities

This section describes the utilities available for Contrail RBAC.

- Utility: [rbacutil.py on page 45](#)
- Utility: [chmod2.py on page 45](#)

Utility: rbacutil.py

Use **rbacutil.py** to manage **api-access-list** rules. It allows adding, removing, and viewing of rules.

Read RBAC rule-set using UUID or FQN

To read an RBAC rule-set using FQN domain/project:

```
python /opt/contrail/utils/rbacutil.py --uuid '$ABC123' --op read
python /opt/contrail/utils/rbacutil.py --name
'default-domain:default-api-access-list' --op read
```

Create RBAC rule-set using FQN domain/project

To create the RBAC rule-set, using UUID or FQN:

```
python /opt/contrail/utils/rbacutil.py --fq_name
'default-domain:api-access-list' --op create
```

Delete RBAC group using FQN or UUID

To delete an RBAC group using FQN or UUID:

```
python /opt/contrail/utils/rbacutil.py --name 'default-domain:api-access-list'
--op delete
python /opt/contrail/utils/rbacutil.py --uuid $ABC123 --op delete
```

Add rule to existing RBAC group

To add a rule to an existing RBAC group:

```
python /opt/contrail/utils/rbacutil.py --uuid <uuid> --rule "*" Member:R" --op
add-rule
python /opt/contrail/utils/rbacutil.py --uuid <uuid> --rule "useragent-kv
*:CRUD" --op add-rule
```

Delete rule from RBAC group - specify rule number or exact rule

To delete a rule from an RBAC group, and specify a rule number or exact rule:

```
python /opt/contrail/utils/rbacutil.py --uuid <uuid> --rule 2 --op del-rule
python /opt/contrail/utils/rbacutil.py --uuid <uuid> --rule "useragent-kv
*:CRUD" --op del-rule
```

Utility: chmod2.py

The utility **chmod2.py** enables updating object permissions, including:

- Ownership—Specify a new owner tenant UUID.
- Enable/disable sharing with other tenants—Specify the tenants.
- Enable/disable sharing with world—Specify permissions.

Upgrading from Previous Releases

The **multi_tenancy** parameter is deprecated, starting with Contrail 3.1. The parameter should be removed from the configuration. Instead, use the **aaa_mode** parameter for RBAC to take effect.

If the **multi_tenancy** parameter is not removed, the **aaa-mode** setting is ignored.

Configuring RBAC Using the Contrail User Interface

To use the Contrail UI with RBAC:

1. Set the **aaa_mode** to **no_auth**.

```
/etc/contrail/contrail-analytics-api.conf
```

```
aaa_mode = no-auth
```

2. Restart the **analytics-api** service.

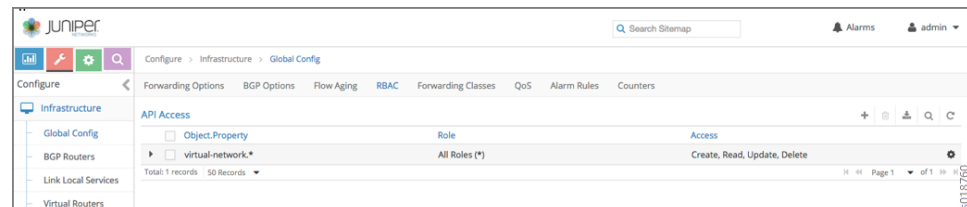
```
service contrail-analytics-api restart
```

You can use the Contrail UI to configure RBAC at both the API level and the object level. API level access control can be configured at the global, domain, and project levels. Object level access is available from most of the create or edit screens in the Contrail UI.

Configuring RBAC at the Global Level

To configure RBAC at the global level, navigate to **Configure > Infrastructure > Global Config > RBAC**, see [Figure 8 on page 46](#).

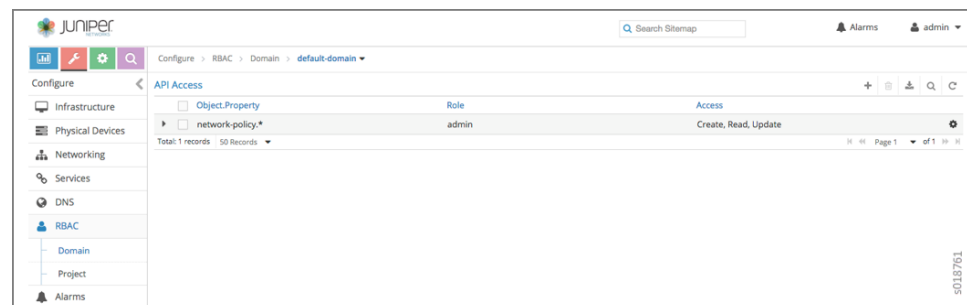
Figure 8: RBAC Global Level



Configuring RBAC at the Domain Level

To configure RBAC at the domain level, navigate to **Configure > RBAC > Domain**, see [Figure 9 on page 46](#).

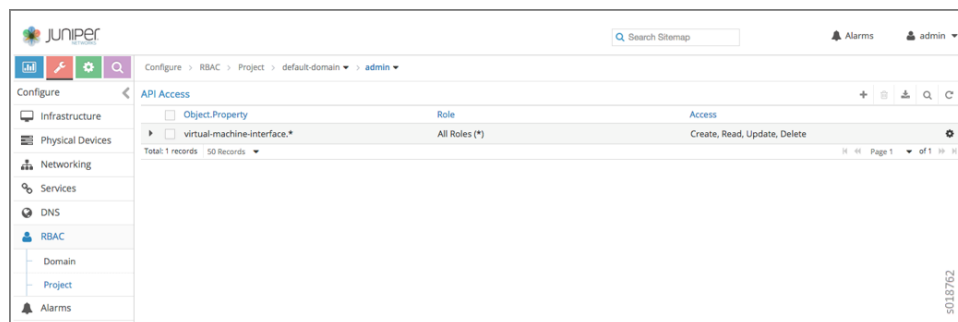
Figure 9: RBAC Domain Level



Configuring RBAC at the Project Level

To configure RBAC at the project level, navigate to **Configure > RBAC > Project**, see [Figure 10 on page 47](#).

Figure 10: RBAC Project Level



Configuring RBAC Details

Configuring RBAC is similar at all of the levels. To add or edit an API access list, navigate to the global, domain, or project page, then click the plus (+) icon to add a list, or click the gear icon to select from Edit, Insert After, or Delete, see [Figure 11 on page 47](#).

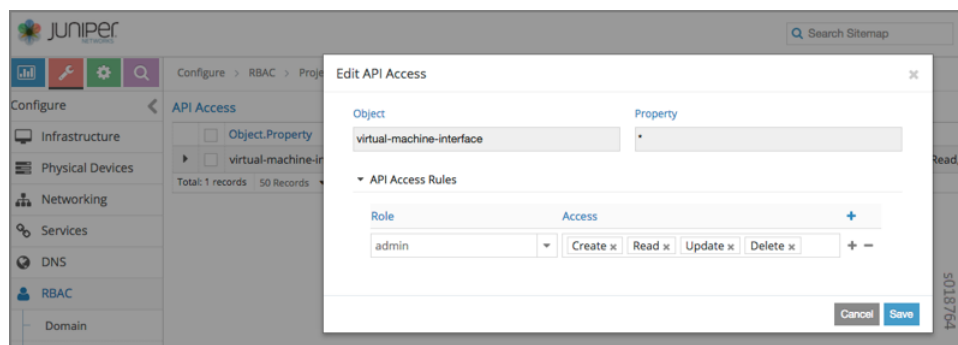
Figure 11: RBAC Details API Access



Creating or Editing API Level Access

Clicking create, edit, or insert after activates the Edit API Access popup window, where you enter the details for the API Access Rules. Enter the user type in the Role field, and use the + icon in the Access field to enter the types of access allowed for the role, including, Create, Read, Update, Delete, and so on, see [Figure 12 on page 47](#).

Figure 12: Edit API Access



Creating or Editing Object Level Access

You can configure fine-grained access control by resource. A **Permissions** tab is available on all create or edit popups for resources. Use the **Permissions** popup to configure owner permissions and global share permissions. You can also share the resource to other tenants by configuring it in the **Share List**, see [Figure 13 on page 48](#).

Figure 13: Edit Object Level Access

The screenshot shows a web-based 'Edit' dialog box with a close button (X) in the top right corner. It features two tabs: 'Network' and 'Permissions', with 'Permissions' currently selected. Under the 'Owner' section, there is a text field containing the UUID 'e5071271c48b432b9ca42572600bf1f6'. Below this, the 'Owner Permissions' section contains three buttons: 'Read x', 'Write x', and 'Refer x'. The 'Global Share Permissions' section has a 'Select Permissions' button. At the bottom, the 'Share List' section is expanded, showing a table with two columns: 'Project' and 'Permissions'. A blue plus sign (+) is located to the right of the table header. On the right side of the dialog, there is a vertical label 's018765'. At the bottom right, there are 'Cancel' and 'Save' buttons.

RBAC Resources

Refer to the *OpenStack Administrator Guide* for additional information about RBAC:

- [Identity API protection with role-based access control \(RBAC\)](#)

CHAPTER 4

Installation and Configuration Scenarios

- [Setting Up and Using a Simple Virtual Gateway with Contrail 4.0 on page 49](#)
- [Configuring MD5 Authentication for BGP Sessions on page 59](#)
- [Configuring the Data Plane Development Kit \(DPDK\) Integrated with Contrail vRouter on page 60](#)
- [Configuring Single Root I/O Virtualization \(SR-IOV\) on page 62](#)
- [Configuring Virtual Networks for Hub-and-Spoke Topology on page 67](#)
- [Configuring Transport Layer Security-Based XMPP in Contrail on page 72](#)
- [Configuring Graceful Restart and Long-lived Graceful Restart on page 74](#)

Setting Up and Using a Simple Virtual Gateway with Contrail 4.0

- [Introduction to the Simple Gateway on page 49](#)
- [How the Simple Gateway Works on page 50](#)
- [Setup Without Simple Gateway on page 50](#)
- [Setup With a Simple Gateway on page 51](#)
- [Simple Gateway Configuration Features on page 52](#)
- [Packet Flows with the Simple Gateway on page 53](#)
- [Packet Flow Process From the Virtual Network to the Public Network on page 53](#)
- [Packet Flow Process From the Public Network to the Virtual Network on page 54](#)
- [Methods for Configuring the Simple Gateway on page 55](#)
- [Using the vRouter Configuration File to Configure the Simple Gateway on page 55](#)
- [Using Thrift Messages to Dynamically Configure the Simple Gateway on page 55](#)
- [Common Issues with Simple Gateway Configuration on page 58](#)

Introduction to the Simple Gateway

Every virtual network has a routing instance associated with it. The routing instance defines the network connectivity for the virtual machines in the virtual network. By default, the routing instance contains routes only for virtual machines spawned within the virtual network. Connectivity between virtual networks is controlled by defining network policies.

The public network is the IP fabric or the external networks across the IP fabric. The virtual networks do not have access to the public network, and a gateway is used to provide connectivity to the public network from a virtual network. In traditional deployments, a routing device such as a Juniper Networks MX Series router can act as a gateway.

The simple virtual gateway for Contrail is a restricted implementation of a gateway that can be used for experimental purposes. The simple gateway provides the Contrail virtual networks with access to the public network, and is represented as **vgw**.

How the Simple Gateway Works

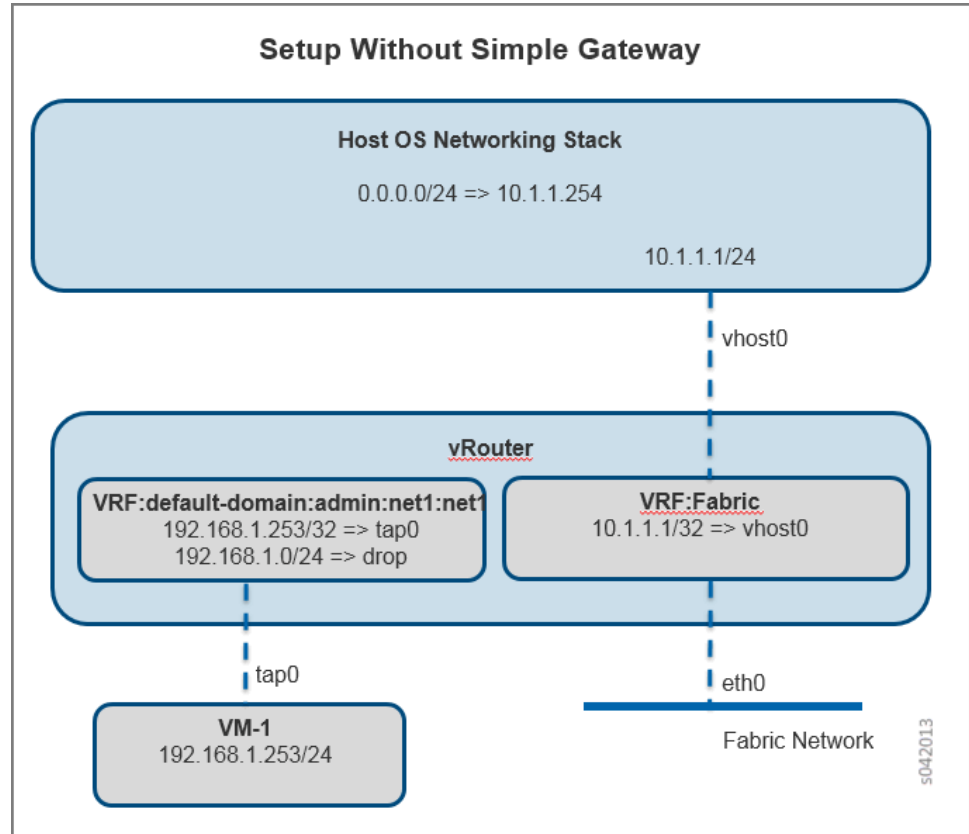
The following sections illustrate how the simple gateway works, first, by showing a virtual network setup with no simple gateway, then illustrating the same setup with a simple gateway configured.

Setup Without Simple Gateway

The following shows a virtual network setup when the simple gateway is not configured.

- A virtual network, **default-domain:admin:net1**, is configured with the subnet 192.168.1.0/24.
- The routing instance **default-domain:admin:net1:net1** is associated with the **default-domain:admin:net1** virtual network .
- A virtual machine with the 192.168.1.253 IP address is spawned in net1.
- A virtual machine is spawned on compute server 1.
- An interface, **vhost0**, is in the host OS of server 1 and is assigned the 10.1.1.1/24 IP address.

- The **vhost0** interface is added to the vRouter in the routing instance fabric.
- The simple gateway is not configured.



Setup With a Simple Gateway

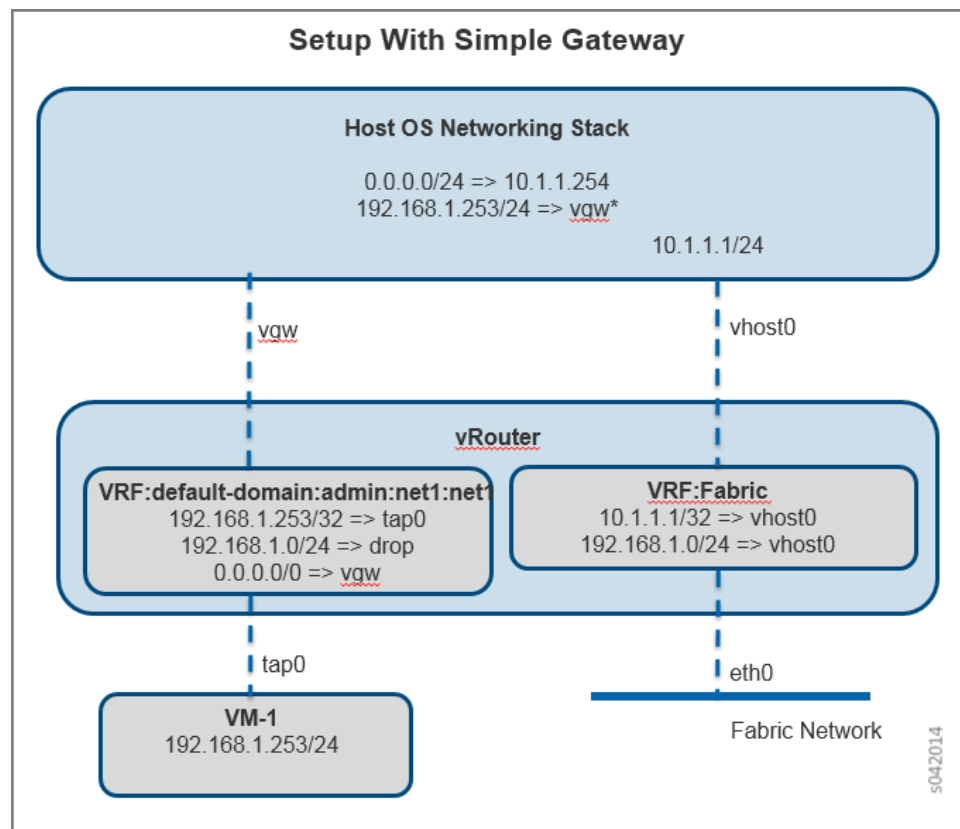
Figure 14 on page 52 shows a virtual network setup with the simple gateway configured for the **default-domain:admin:net1** virtual network.

The simple gateway configuration uses a gateway interface (vgw) to provide connectivity between the fabric routing instance and the **default-domain:admin:net1** virtual network.

Figure 14 on page 52 shows the packet flows between the fabric VRF and the **default-domain:admin:net1** virtual network.

In the diagram, routes marked with (*) are added by the simple gateway feature.

Figure 14: Virtual Network Setup With a Simple Gateway



Simple Gateway Configuration Features

The simple gateway configuration has the following features.

- The simple gateway is configured for the **default-domain:admin:net1** virtual network.
 - The **vgw** gateway interface provides connectivity between the routing instance **default-domain:admin:net1:net1** and the fabric.
 - An IP address is not configured for the **vgw** gateway interface.
- The host OS is configured with the following:
 - Two INET interfaces are added to the host OS: **vgw** and **vhost0**
 - The host OS is not aware of the routing instances, so the **vgw** and **vhost0** interfaces are part of the same routing instance in the host OS.
 - The simple gateway adds the **192.168.1.0/24** route, pointing to the **vgw** interface, and that setup is added to the host OS. This route ensures that any packet destined to the virtual machine is sent to the vRouter on the **vgw** interface.
- The vRouter is configured with the following:

- The routing instance named **Fabric** is created for the fabric network.
- The interface **vhost0** is added to the routing instance **Fabric**.
- The interface **eth0**, which is connected to the fabric network, is added to the routing instance named **Fabric**.
- The simple gateway adds the **192.168.1.0/24** route to the **vhost0** interface. Consequently, packets destined to the **default-domain:admin:net1** virtual network are sent to the host OS.
- The **default-domain:admin:net1:net1** routing instance is created for the **default-domain:admin:net1** virtual network.
 - The **vgw** interface is added to the **default-domain:admin:net1:net1** routing instance.
 - The simple gateway adds a default route (**0.0.0.0/0**) that points to the **vgw** interface. Packets in the **default-domain:admin:net1:net** routing instance that match this route are sent to the host OS on the **vgw** interface. The host OS routes the packets to the fabric network over the **vhost0** interface.

Simple Gateway Restrictions

The following are restrictions of the simple gateway:

- A single compute node can have the simple gateway configured for multiple virtual networks, however, there cannot be overlapping subnets. The host OS does not support routing instances. Therefore, all gateway interfaces in the host OS are in the same routing instance and the subnets in the virtual networks must not overlap.
- Each virtual network can have a single simple gateway interface. ECMP is not supported.

Packet Flows with the Simple Gateway

The following sections describe the packet flow process when the simple gateway is configured on a Contrail system.

First, the packet flow process from the virtual network to the public network is described. Next, the packet flow process from the public network to the virtual network is described.

Packet Flow Process From the Virtual Network to the Public Network

The following describes the procedure used to move a packet from the virtual network (net1) to the public network.

1. A packet with a source IP address of **192.168.1.253** and a destination IP address of **10.1.1.253** comes from a virtual machine and is received by the vRouter on the **tap0** interface.
2. The **tap0** interface is in the **default-domain:admin:net1:net1** routing instance.
3. The route lookup for **10.1.1.253** in the **default-domain:admin:net1:net1** routing instance finds the default route pointing to the tap interface named **vgw**.

4. The vRouter transmits the packet toward the **vgw** interface and it is received by the networking stack of the host OS.
5. The host OS performs forwarding based on its routing table and forwards the packet on the **vhost0** interface.
6. Packets transmitted on the **vhost0** interface are received by the vRouter.
7. The **vhost0** interface is added to the **Fabric** routing instance.
8. The routing table for 10.1.1.253 in the **Fabric** routing instance indicates that the packet is to be transmitted on the **eth0** interface.
9. The vRouter transmits the packet on the **eth0** interface.
10. The 10.1.1.253 host on the **Fabric** routing instance receives the packet.

Packet Flow Process From the Public Network to the Virtual Network

The following describes the procedure used to move a packet from the public network to the virtual network (net1).

1. A packet with a source IP address of **10.1.1.253** and a destination IP address of **192.168.1.253** coming from the public network is received on the **eth0** interface.
2. The **tap0** interface is in the **default-domain:admin:net1:net1** routing instance.
3. The vRouter receives the packet from the **eth0** interface in the **Fabric** routing instance.
4. The route lookup for 192.168.1.253 in the **Fabric** routing instance points to the interface **vhost0**.
5. The vRouter transmits the packet on the **vhost0** interface and it is received by the networking stack of the host OS.
6. The host OS performs forwarding according to its routing table and forwards the packet on the **vgw** interface.
7. The vRouter receives the packet on the **vgw** interface into the routing instance **default-domain:admin:net1:net1**.
8. The route lookup for 192.168.1.253 in the **default-domain:admin:net1:net1** routing instance points to the **tap0** interface.

9. The vRouter transmits the packet on the **tap0** interface.
10. The virtual machine receives the packet destined to 192.168.1.253.

Methods for Configuring the Simple Gateway

There are different methods that can be used to configure the simple gateway. Each of the methods is described in the following sections.

Using the vRouter Configuration File to Configure the Simple Gateway

Another way to enable a simple gateway is to configure one or more **vgw** interfaces within the **contrail-vrouter-agent.conf** file.

Any changes made in this file for simple gateway configuration are implemented upon the next restart of the vRouter agent. To configure the simple gateway in the **contrail-vrouter-agent.conf** file, each simple gateway interface uses the following parameters:

- **interface=vgwxx**— Simple gateway interface name.
- **routing_instance=default-domain:admin:public xx:public xx**— Name of the routing instance for which the simple gateway is being configured.
- **ip_block=1.1.1.0/24**— List of the subnet addresses allocated for the virtual network. Routes within this subnet are added to both the host OS and routing instance for the fabric instance. Represent multiple subnets in the list by separating each with a space.
- **routes=10.10.10.1/24 11.11.11.1/24**— List of subnets in the public network that are reachable from the virtual network. Routes within this subnet are added to the routing instance configured for the **vgw** interface. Represent multiple subnets in the list by separating each with a space.

Using Thrift Messages to Dynamically Configure the Simple Gateway

Another way to configure the simple gateway is to dynamically send create and delete thrift messages to the vrouter agent.

Starting with Contrail Release 1.10 and greater, the following thrift messages are available:

- **AddVirtualGateway**—add a virtual gateway
- **DeleteVirtualGateway**—delete a virtual gateway
- **ConnectForVirtualGateway**—allows audit of the virtual gateway configuration by stateful clients. Upon a new **ConnectForVirtualGateway** request, one minute is allowed for the configuration to be redone. Any older virtual gateway configuration remaining after this time is deleted.
- [How to Dynamically Create a Virtual Gateway on page 56](#)
- [How to Dynamically Delete a Virtual Gateway on page 56](#)
- [Using Devstack to Configure the Simple Gateway on page 57](#)

How to Dynamically Create a Virtual Gateway

To dynamically create a simple virtual gateway, you run a script on the compute node where the virtual gateway is being created.

When run, the script does the following:

1. Enables forwarding on the node.
2. Creates the required interface.
3. Adds the interface to the vRouter.
4. Adds required routes to the host OS.
5. Sends the **AddVirtualGateway** thrift message to the vRouter agent telling it to create the virtual gateway.

Example: Dynamically Create a Virtual Gateway

The following procedure dynamically creates the **vgw1** interface, with **20.30.40.0/24** and **30.40.50.0/24** subnets in the **default-domain:admin:vn1:vn1** VRF.

1. Set the **PYTHONPATH** variable to the location of the **InstanceService.py** and **types.py** files, for example:

```
export  
PYTHONPATH=/usr/lib/python2.7/dist-packages/nova_contrail_vif/gen_py/instance_service  
  
export  
PYTHONPATH=/usr/lib/python2.6/site-packages/contrail_vrouter_api/gen_py/instance_service
```
2. Run the virtual gateway **provision** command with the **oper create** option.
Use the **subnets** option to specify the subnets defined for virtual network **vn1**.
Use the **routes** option to specify the routes in the public network that are injected into **vn1**.
In the following example, the virtual machines in **vn1** can access subnets **8.8.8.0/24** and **9.9.9.0/24** in the public network:

```
python /opt/contrail/utils/provision_vgw_interface.py --oper create --interface vgw1  
--subnets 20.30.40.0/24 30.40.50.0/24 --routes 8.8.8.0/24 9.9.9.0/24 --vrf  
default-domain:admin:vn1:vn1
```

How to Dynamically Delete a Virtual Gateway

To dynamically delete a virtual gateway, run a script on the compute node where the virtual gateway is.

When run, the script does the following:

1. Sends the **DeleteVirtualGateway** thrift message to the vRouter agent. Tell it to delete the virtual gateway.
2. Deletes the virtual gateway interface from the vRouter.
3. Deletes the virtual gateway routes that were added in the host OS when the virtual gateway was created.

Example: Dynamically Create a Virtual Gateway

The following procedure dynamically deletes the **vgw1** interface. It also deletes the 20.30.40.0/24 and 30.40.50.0/24 subnets in the **default-domain:admin:vn1:vn1** VRF .

1. Set the **PYTHONPATH** variable to the location of the **InstanceService.py** and **types.py** files, for example:


```
export
PYTHONPATH=/usr/lib/python2.7/dist-packages/nova_contrail_vif/gen_py/instance_service

export
PYTHONPATH=/usr/lib/python2.6/site-packages/contrail_vrouter_api/gen_py/instance_service
```
2. Run the virtual gateway **provision** command with the **oper delete** option.


```
python /opt/contrail/utils/provision_vgw_interface.py --oper delete --interface vgw1
--subnets 20.30.40.0/24 30.40.50.0/24 --routes 8.8.8.0/24 9.9.9.0/24
```
3. (optional) If you are using a stateful client, send the **ConnectForVirtualGateway** thrift message to the vRouter agent when the client starts.



NOTE: If the vRouter agent restarts or if the compute node reboots, it is expected that the client reconfigures again.

Using Devstack to Configure the Simple Gateway

Another way to configure the simple gateway is to set configuration parameters in the devstack **localrc** file.

The following parameters are available:

- **CONTRAIL_VGW_PUBLIC_NETWORK** — The name of the routing instance for which the simple gateway is being configured.
- **CONTRAIL_VGW_PUBLIC_SUBNET** — A list of subnet addresses allocated for the virtual network. Routes containing these addresses are added to both the host OS and the routing instance for the fabric. List multiple subnets by separating each with a space.
- **CONTRAIL_VGW_INTERFACE** — A list of subnets in the public network that are reachable from the virtual network. Routes containing these subnets are added to the routing instance configured for the simple gateway. List multiple subnets by separating each with a space.

This method can only add the default route **0.0.0.0/0** into the routing instance specified in the **CONTRAIL_VGW_PUBLIC_NETWORK** option.

**Example: Devstack
Configuration for
Simple Gateway**

Add the following lines in the **localrc** file for **stack.sh**:

```
CONTRAIL_VGW_INTERFACE=vgw1  
  
CONTRAIL_VGW_PUBLIC_SUBNET=192.168.1.0/24  
  
CONTRAIL_VGW_PUBLIC_NETWORK=default-domain:admin:net1:net1
```



NOTE: This method can only add the **0.0.0.0/0** default route into the routing instance specified in the **CONTRAIL_VGW_PUBLIC_NETWORK** option.

Common Issues with Simple Gateway Configuration

The following are common problems you might encounter when configuring a simple gateway.

- Packets from the external network are not reaching the compute node.

The devices in the fabric network must be configured with static routes for the IP addresses defined in the public subnet (192.168.1.0/24 in the example) to reach the compute node that is running as a simple gateway.

- Packets are reaching the compute node, but are not routed from the host OS to the virtual machine.

Check to see if the **firewall_driver** in the **/etc/nova/nova.conf** file is set to **nova.virt.libvirt.firewall.IptablesFirewallDriver**, which enables IPTables. IPTables can discard packets.

Resolutions include disabling IPTables during runtime or setting the **firewall_driver** in the **localrc** file: **LIBVIRT_FIREWALL_DRIVER=nova.virt.firewall.NoopFirewallDriver**

Configuring MD5 Authentication for BGP Sessions

Contrail supports MD5 authentication for BGP peering based on RFC 2385.

This option allows BGP to protect itself against the introduction of spoofed TCP segments into the connection stream. Both of the BGP peers must be configured with the same MD5 key. Once configured, each BGP peer adds a 16-byte MD5 digest to the TCP header of every segment that it sends. This digest is produced by applying the MD5 algorithm on various parts of the TCP segment. Upon receiving a signed segment, the receiver validates it by calculating its own digest from the same data (using its own key) and compares the two digests. For valid segments, the comparison is successful since both sides know the key.

The following are ways to enable BGP MD5 authentication and set the keys on the Contrail node.

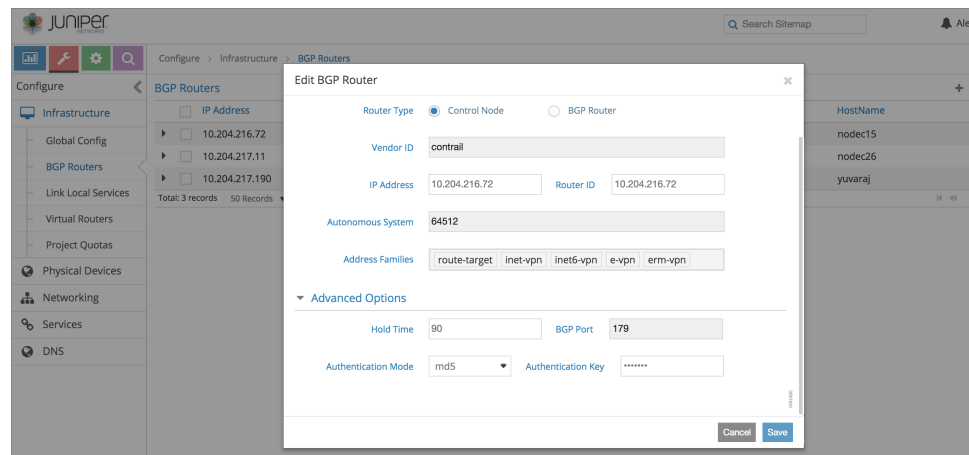
1. If the **md5** key is not included in the provisioning, and the node is already provisioned, you can run the following script with an argument for md5:

```
contrail-controller/src/config/utlils/provision_control.py
```

```
host@<your_node>:/opt/contrail/utlils# python provision_control.py --host_name
<host_name> --host_ip <host_ip> --router_asn <asn> --api_server_ip <api_ip>
--api_server_port <api_port> --oper add --md5 "juniper" --admin_user admin
--admin_password <password> --admin_tenant_name admin
```

2. You can also use the web user interface to configure MD5.
 - a. Connect to the node's IP address at port 8080 (<node_ip>:8080) and select **Configure->Infrastructure->BGP Routers**. As shown in [Figure 15 on page 59](#), a list of BGP peers is displayed.

Figure 15: Edit BGP Router Window



- b. For a BGP peer, click on the gear icon on the right hand side of the peer entry. Then click **Edit**. This displays the Edit BGP Router dialog box.

- c. Scroll down the window and select **Advanced Options**.
- d. Configure the MD5 authentication by selecting **Authentication Mode>MD5** and entering the **Authentication Key** value.

- Related Documentation**
- [Creating a Virtual Network with Juniper Networks Contrail on page 296](#)
 - [Creating a Virtual Network with OpenStack Contrail on page 299](#)

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

- [DPDK Support in Contrail on page 60](#)
- [Preparing the server.json File for Provisioning a Contrail 4.0 Cluster Node with DPDK on page 60](#)
- [Creating a Flavor for DPDK in OpenStack Kilo on page 62](#)

DPDK Support in Contrail

Contrail 3.0 and later supports the Data Plane Development Kit (DPDK) on Ubuntu systems, only.

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.

When using DPDK with Contrail 4.0, one or more Contrail compute nodes are provisioned with DPDK during installation. An entry in the server configuration file specifies which nodes are to be configured to use the DPDK vRouter mode instead of the regular kernel mode. This allows for a mixed setup, where different nodes use different modes of the vRouter.

Upon installation, when a Contrail compute node is provisioned with DPDK, the server configuration file specifies which physical interface(s) to use, how many CPU cores to use for forwarding packets, and the number of huge pages to allocate for DPDK.

Preparing the server.json File for Provisioning a Contrail 4.0 Cluster Node with DPDK

The **server.json** file is used with Server Manager or SM-Lite 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 **server.json** with a dictionary entry, along with the percentage of memory for DPDK huge pages and the CPUs to be used, as in the following example:

```

{
  "server" : [{
    "parameters" : {
      "provision": {
        "contrail_4": {
          "core_mask": "0x3f",
          "huge_pages": "50"
        }
      }
    }
  }]
}

```

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.

- **coremask**—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”.

Supported formats include:

- Hexadecimal (for example, 0xf)
- Comma-separated list of CPUs (1,2,4...)
- Dash-separated range of CPUs (for example, 1-4)

The server configuration file is configured prior to the installation of Contrail.

Use the standard Contrail installation procedure, and upon completion, your cluster with specified nodes using the DPDK vRouter implementation is ready to use.

Support for Multiple UIO Drivers

Support is available for optionally specifying the userspace IO (UIO) driver to use in a DPDK-enabled compute node.

Specify UIO in the **dppdk** section of the server configuration file, by using the optional attribute **uio_driver**:

```

{
  "server" : [{
    "parameters" : {
      "provision": {
        "contrail_4": {
          "core_mask": "0x3f",
          "huge_pages": "50",
          "uio_driver" : "igb_uio"
        }
      }
    }
  }]
}

```

```
    }  
  }  
}  
}]  
}
```

The supported values for **uio_driver** include:

- **igb_uio**—specify that the igb_uio module from the DPDK library should be used.
- **vfio-pci**—specify that the vfio module in the Linux kernel should be used instead of uio, which protects memory accesses 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 that is built into the Linux kernel should be used. This option does not support the use of SR-IOV VFs.

If the **uio_driver** is not specified in the server configuration file, **igb_uio** is used by default.

Creating a Flavor for DPDK in OpenStack Kilo

OpenStack Kilo has a feature called flavors, which are virtual hardware templates that define sizes for RAM, disk, and so on. Contrail 3.0 and later supports the OpenStack Kilo flavor that specifies that a VM should 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.

```
$nova flavor-key m1.large set 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.

Related Documentation

- [Configuring Single Root I/O Virtualization \(SR-IOV\) on page 62](#)
- DPDK official page: <http://www.dpdk.org>

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

- [Overview: Configuring SR-IOV on page 63](#)
- [Configuring SR-IOV Using JSON on page 63](#)
- [Preparing the testbed.py File for Provisioning a Contrail Cluster with SR-IOV on page 63](#)
- [Enabling ASPM in BIOS on page 64](#)
- [Configuring SR-IOV Features Without env.sriov in testbed.py on page 64](#)
- [Launching SR-IOV Virtual Machines on page 65](#)

Overview: Configuring SR-IOV

Contrail 3.0 and later supports single root I/O virtualization (SR-IOV) on Ubuntu systems only.

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 system.

Configuring SR-IOV Using JSON

If you are provisioning your system by using Server Manager or SM-Lite, you can configure SR-IOV within the **server.json** file.

```

"parameters" : {
  "provision": {
    "contrail_4": {
      "sriov": {
        "p5p1": {
          "VF": 7,
          "physnets": [
            "physnet1"
          ]
        }
      }
    }
  },
}
```

Preparing the testbed.py File for Provisioning a Contrail Cluster with SR-IOV

The **testbed.py** file is a Python file that is configured to specify all of the options necessary for the installation of a Contrail cluster. The testbed.py file can only be used with Contrail releases through 3.x.x, or with SM-Lite provisioning with Contrail 4.0.

An **env.sriov** entry in the **testbed.py** file is used to specify which NIC interface will be used to launch SR-IOV virtual machines (VMs).

Each VM to be configured as SR-IOV must be listed in the **env.sriov** entry, along with the number of virtual functions to be used and the physical network details, as in the following example:

```

env.sriov = {
  b7s36 :[ {'interface' : 'p6p1', 'VF' : 7, 'physnets' : ['physnet1',
'physnet2']}],
  b7s37 :[ {'interface' : 'p6p1', 'VF' : 7, 'physnets' : ['physnet1',
'physnet3']}],
}
```

The following are required entries for each VM specified in the **env.sriov**:

- **interface**—Specify the server NIC where SR-IOV VMs will be launched.
- **VF**—Specify the number of virtual functions to be configured.
- **physnets**—Specify the name of the physical networks to be used by each VM.

The **testbed.py** file is configured prior to the installation of Contrail.

Use the standard Contrail installation procedure with fab tools, and upon completion, your cluster is ready to be enabled to launch SR-IOV VMs with specified NICs.

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 Features Without env.sriov in testbed.py

If you are enabling SR-IOV on a system where **testbed.py** with **env.sriov** was NOT used to install, you must also complete the following:

1. Enable the Intel Input-Output Memory Management Unit (IOMMU) on Linux, by doing the following:
 - a. Make sure the IOMMU is turned on, using the following option line in **/etc/default/grub**:
GRUB_CMDLINE_LINUX_DEFAULT="nomdmonddf nomdmonisw intel_iommu=on"
 - b. Enter the command **update-grub/sys**.
 - c. Reboot the compute node.
2. Enable the required number of VFs on the selected NIC.
The following example enables seven VFs on the **eth0** interface. Check the configuration by using the command **lspci -nn** or **ip link**.

```
echo '7' > /sys/class/net/eth0/device/sriov_numvfs
```

3. In the Nova config file on the controller, configure the names of the physical networks whose VMs can use interface VFs. The following example enables the VMs attached to **"physnet1"** to use the VFs of **"eth0"**:

```
/etc/nova/nova.conf
[default]
pci_passthrough_whitelist = { "devname": "eth0", "physical_network":
"physnet1"}
```

4. Reboot Nova compute.

service nova-compute restart

5. In the Nova config file, 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

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 UI or the CLI. Both methods are included.

- [Using the Contrail UI to Enable and Launch an SR-IOV Virtual Machine on page 65](#)
- [Using the CLI to Enable and Launch SR-IOV Virtual Machines on page 66](#)

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

To use the Contrail 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 16 on page 65](#).

Figure 16: Edit Network

The screenshot shows a web-based 'Edit Network' form. At the top is a title bar with the text 'Edit Network' and a close icon. Below this, there are two main sections. The first section is labeled 'Provider Network' and contains a checkbox that is currently checked. The second section is labeled 'Physical Network' and contains a text input field with the placeholder text 'Network Name'. To the right of this input field is a 'VLAN' section, which also contains a text input field with the value '1 - 4094'. On the far right side of the form, there is a vertical label 's018550'.

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 17 on page 66](#).

Figure 17: 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

Cancel Save

S018551

- 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:

- 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
```

- Create a subnet in **vn1**.

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

- 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
```

- 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>
```

- 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](#)

Configuring Virtual Networks for Hub-and-Spoke Topology

As of Contrail Release 3.0, 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.

- [Route Targets for Virtual Networks in Hub-and-Spoke Topology on page 67](#)
- [Example: Configuring Hub-and-Spoke Virtual Networks on page 68](#)
- [Troubleshooting Hub-and-Spoke Topology on page 68](#)

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: Configuring Hub-and-Spoke Virtual Networks

The following example uses a script to configure the hub-and-spoke virtual networks.

In the example, the “**hub-vn**” is configured as a hub virtual network, with the import route target of “**target:1:1**” and the export route target of “**target:1:2**”. The “**spoke-vn***” is configured as a spoke virtual network, with the import route target of “**target:1:2**” and the export route target of “**target:1:1**”.

The **spoke-rt** is “**target:1:1**” and the **hub-rt** is “**target:1:2**”, consequently, the “**hub-vn**” imports “**spoke-rt**” and exports “**hub-rt**”, and the **spoke-vn** imports “**hub-rt**” and exports “**spoke-rt**”.

Using vnc-api to Configure Hub-and-Spoke Topology Example

```
from vnc_api.vnc_api import *
lib = VncApi("admin", "<password>", "admin", "<ip address>", "8082")
vn=lib.virtual_network_read(fq_name=["default-domain", "admin", "hub-vn"])
vn.set_import_route_target_list(RouteTargetList(["target:1:1"]))
vn.set_export_route_target_list(RouteTargetList(["target:1:2"]))
lib.virtual_network_update(vn)

vn=lib.virtual_network_read(fq_name=["default-domain", "admin", "spoke-vn1"])
vn.set_import_route_target_list(RouteTargetList(["target:1:2"]))
vn.set_export_route_target_list(RouteTargetList(["target:1:1"]))
lib.virtual_network_update(vn)

vn=lib.virtual_network_read(fq_name=["default-domain", "admin", "spoke-vn2"])
vn.set_import_route_target_list(RouteTargetList(["target:1:2"]))
vn.set_export_route_target_list(RouteTargetList(["target:1:1"]))
lib.virtual_network_update(vn)

vn=lib.virtual_network_read(fq_name=["default-domain", "admin", "spoke-vn3"])
vn.set_import_route_target_list(RouteTargetList(["target:1:2"]))
vn.set_export_route_target_list(RouteTargetList(["target:1:1"]))
lib.virtual_network_update(vn)

vn=lib.virtual_network_read(fq_name=["default-domain", "admin", "spoke-vn4"])
vn.set_import_route_target_list(RouteTargetList(["target:1:2"]))
vn.set_export_route_target_list(RouteTargetList(["target:1:1"]))
lib.virtual_network_update(vn)
```

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/utls# 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 18 on page 72 shows the import and export targets for **hub-vn** and **spoke-vns**, by invoking **contrail-control-introspect**.

Figure 18: Contrail Introspect

The screenshot shows the Contrail Introspect web interface. The browser address bar displays 'nodec13.englab.juniper.net:8083/ShowRoutingInstanceSummary?search_string='. The page title is 'Contrail'. Below the title, there is a search bar and a table of routing instances. The table has columns: name, virtual_network, vn_index, vxlan_id, import_target, and export_target. The table lists several instances, including 'default-domain:admin:hub-vn:hub-vn' and 'default-domain:admin:spoke-vn1:spoke-vn1'. The right side of the interface shows a sidebar with a search bar and a list of 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	export_target
default-domain:admin:spoke-vn1:spoke-vn1	default-domain:admin:spoke-vn1	11	0	import_target	export_target
default-domain:admin:spoke-vn2:spoke-vn2	default-domain:admin:spoke-vn2	12	0	import_target	export_target
default-domain:admin:spoke-vn3:spoke-vn3	default-domain:admin:spoke-vn3	14	0	import_target	export_target
default-domain:admin:spoke-vn4:spoke-vn4	default-domain:admin:spoke-vn4	13	0	import_target	export_target

Configuring Transport Layer Security-Based XMPP in Contrail

Overview: TLS-Based XMPP

Starting with Contrail 3.0, Transport Layer Security (TLS)-based XMPP can be used to secure all Extensible Messaging and Presence Protocol (XMPP)-based communication that occurs in the Contrail environment.

Secure XMPP is based on *RFC 6120, Extensible Messaging and Presence Protocol (XMPP): Core*.

TLS XMPP in Contrail

In the Contrail environment, the Transport Layer Security (TLS) protocol is used for certificate exchange, mutual authentication, and negotiating ciphers to secure the stream from potential tampering and eavesdropping.

The RFC 6120 highlights a basic stream message exchange format for TLS negotiation between an XMPP server and an XMPP client.



NOTE: Simple Authentication and Security Layer (SASL) authentication is not supported in the Contrail environment.

Configuring XMPP Client and Server in Contrail

In the Contrail environment, XMPP based communications are used in client and server exchanges, between the compute node (as the XMPP client), and:

- the control node (as the XMPP server)
- the DNS server (as the XMPP server)

Configuring Control Node for XMPP Server

To enable secure XMPP, the following parameters are configured at the XMPP server.

On the control node, enable the parameters in the configuration file:

/etc/contrail/contrail-control.conf.

Parameter	Description	Default
xmpp_server_cert	Path to the node's public certificate	/etc/contrail/ssl/certs/server.pem
xmpp_server_key	Path to server's or node's private key	/etc/contrail/ssl/private/server-privkey.pem
xmpp_ca_cert	Path to CA certificate	/etc/contrail/ssl/certs/ca-cert.pem
xmpp_auth_enable=true	Enables SSL based XMPP	Default is set to false, XMPP is disabled. NOTE: The keyword true is case sensitive.

Configuring DNS Server for XMPP Server

To enable secure XMPP, the following parameters are configured at the XMPP DNS server.

On the DNS server control node, enable the parameters in the configuration file:

/etc/contrail/contrail-control.conf

Parameter	Description	Default
xmpp_server_cert	Path to the node's public certificate	/etc/contrail/ssl/certs/server.pem
xmpp_server_key	Path to server's/node's private key	/etc/contrail/ssl/certs/server-privkey.pem
xmpp_ca_cert	Path to CA certificate	/etc/contrail/ssl/certs/ca-cert.pem
xmpp_dns_auth_enable=true	Enables SSL based XMPP	Default is set to false, XMPP is disabled. NOTE: The keyword true is case sensitive.

Configuring Control Node for XMPP Client

To enable secure XMPP, the following parameters are configured at the XMPP client.

On the compute node, enable the parameters in the configuration file:

/etc/contrail/contrail-vrouter-agent.conf

Parameter	Description	Default
xmpp_server_cert	Path to the node's public certificate	/etc/contrail/ssl/certs/server.pem

Parameter	Description	Default
<code>xmpp_server_key</code>	Path to server's/node's private key	<code>/etc/contrail/ssl/private/server-privkey.pem</code>
<code>xmpp_ca_cert</code>	Path to CA certificate	<code>/etc/contrail/ssl/certs/ca-cert.pem</code>
<code>xmpp_auth_enable=true</code> <code>xmpp_dns_auth_enable=true</code>	Enables SSL based XMPP	Default is set to false, XMPP is disabled. NOTE: The keyword true is case sensitive.

Configuring Graceful Restart and Long-lived Graceful Restart

Starting with Contrail Release 3.2, graceful restart and long-lived graceful restart BGP helper modes are supported for the Contrail control node. Release 4.1 introduces support for the XMPP helper mode as well.

- [Application of Graceful Restart and Long-lived Graceful Restart on page 74](#)
- [BGP Graceful Restart Helper Mode on page 75](#)
- [Feature Highlights on page 75](#)
- [XMPP Helper Mode on page 75](#)
- [Configuration Parameters on page 76](#)
- [Cautions for Graceful Restart on page 77](#)
- [Configuring Graceful Restart with the Contrail User Interface on page 78](#)

Application of Graceful Restart and Long-lived Graceful Restart

Whenever a BGP peer session is detected as down, all routes learned from the peer are deleted and immediately withdrawn from advertised peers. This causes instantaneous disruption to traffic flowing end-to-end, even when routes kept in the vrouter kernel in the data plane remain intact.

Graceful restart and long-lived graceful restart features can be used to alleviate traffic disruption caused by downs.

When configured, graceful restart features enable existing network traffic to be unaffected if Contrail controller processes go down. The Contrail implementation ensures that if a Contrail control module restarts, it can use graceful restart functionality provided by its BGP peers. Or when the BGP peers restart, Contrail provides a graceful restart helper mode to minimize the impact to the network. The graceful restart features can be used to ensure that traffic is not affected by temporary outage of processes.

Graceful restart is not enabled by default.

With graceful restart features enabled, learned routes are not deleted when sessions go down, and the routes are not withdrawn from the advertised peers. Instead, the routes are kept and marked as 'stale'. Consequently, if sessions come back up and routes are relearned, the overall impact to the network is minimized.

After a certain duration, if a downed session does not come back up, all remaining stale routes are deleted and withdrawn from advertised peers.

The graceful restart and long-lived graceful restart features can be enabled only for BGP peers in Contrail 3.2. Future releases will provide support for XMPP-based peering sessions (agents).

BGP Graceful Restart Helper Mode

The BGP helper mode can be used to minimize routing churn whenever a BGP session flaps. This is especially helpful if the SDN gateway router goes down gracefully, as in an rpd crash or restart on an MX Series Junos device. In that case, the contrail-control can act as a graceful restart helper to the gateway, by retaining the routes learned from the gateway and advertising them to the rest of the network as applicable. In order for this to work, the restarting router (the SDN gateway in this case) must support and be configured with graceful restart for all of the address families used.

The graceful restart helper mode is also supported for BGP-as-a-Service (BGPaaS) clients. When configured, contrail-control can provide a graceful restart or long-lived graceful restart helper mode to a restarting BGPaaS client.

Feature Highlights

The following are highlights of the graceful restart and long-lived graceful restart features.

- Configuring a non-zero restart time enables the ability to advertise graceful restart and long-lived graceful restart capabilities in BGP.
- Configuring helper mode enables the ability for graceful restart and long-lived graceful restart helper modes to retain routes even after sessions go down.
- With graceful restart configured, whenever a session down event is detected and a closing process is triggered, all routes, across all address families, are marked stale. The stale routes are eligible for best-path election for the configured graceful restart time duration.
- When long-lived graceful restart is in effect, stale routes can be retained for a much longer time than that allowed by graceful restart alone. With long-lived graceful restart, route preference is retained and best paths are recomputed. The community marked LLGR_STALE is tagged for stale paths and re-advertised. However, if no long-lived graceful restart community is associated with any received stale route, those routes are not kept, instead, they are deleted.
- After a certain time, if a session comes back up, any remaining stale routes are deleted. If the session does not come back up, all retained stale routes are permanently deleted and withdrawn from the advertised peer.

XMPP Helper Mode

Contrail release 4.1 introduces support for long-lived graceful restart (LLGR) with XMPP helper mode. Graceful restart and long lived graceful restart can be enabled using the Contrail web UI or by using the provision_control script.

The helper modes can also be enabled via schema, and can be disabled selectively in a contrail-control node for BGP or XMPP sessions by configuring **gr_helper_disable** in the **/etc/contrail/contrail-control.conf** configuration file.

Configuration Parameters

Graceful restart parameters are configured in the **global-system-config** of the schema. They can be configured by means of a provisioning script or by using the Contrail Web UI.

Configure a non-zero restart time to advertise for graceful restart and long-lived graceful restart capabilities from peers.

Configure helper mode for graceful restart and long-lived graceful restart to retain routes even after sessions go down.

Configuration parameters include:

- **enable** or **disable** for all graceful restart parameters:
 - **restart-time**
 - **long-lived-restart-time**
 - **end-of-rib-timeout**
- **bgp-helper-enable** to enable graceful restart helper mode for BGP peers in contrail-control
- **xmpp-helper-enable** to enable graceful restart helper mode for XMPP peers (agents) in contrail-control

The following shows configuration by a provision script.

```
/opt/contrail/utils/provision_control.py
--api_server_ip 10.xx.xx.20
--api_server_port 8082
--router_asn 64512
--admin_user admin
--admin_password <password>
--admin_tenant_name admin
--set_graceful_restart_parameters
--graceful_restart_time 60
--long_lived_graceful_restart_time 300
--end_of_rib_timeout 30
--graceful_restart_enable
--graceful_restart_bgp_helper_enable
```

The following are sample parameters:

```
-set_graceful_restart_parameters
--graceful_restart_time 300
--long_lived_graceful_restart_time 60000
--end_of_rib_timeout 30
--graceful_restart_enable
--graceful_restart_bgp_helper_enable
```

When BGP peering with Juniper Networks devices, Junos must also be explicitly configured for graceful restart/long-lived graceful restart, as shown in the following example:

```
set routing-options graceful-restart
set protocols bgp group <a1234> type internal
set protocols bgp group <a1234> local-address 10.xx.xxx.181
set protocols bgp group <a1234> keep all
set protocols bgp group <a1234> family inet-vpn unicast graceful-restart long-lived
  restarter stale-time 20
set protocols bgp group <a1234> family route-target graceful-restart long-lived
  restarter stale-time 20
set protocols bgp group <a1234> graceful-restart restart-time 600
set protocols bgp group <a1234> neighbor 10.xx.xx.20 peer-as 64512
```

The graceful restart helper modes can be enabled in the schema. The helper modes can be disabled selectively in the **contrail-control.conf** for BGP sessions by configuring **gr_helper_disable** in the **/etc/contrail/contrail-control.conf** file.

The following are examples:

```
/usr/bin/openstack-config /etc/contrail/contrail-control.conf DEFAULT
gr_helper_bgp_disable 1
```

```
/usr/bin/openstack-config /etc/contrail/contrail-control.conf DEFAULT
gr_helper_xmpp_disable 1
```

```
service contrail-control restart
```

For more details about graceful restart configuration, see <https://github.com/Juniper/contrail-controller/wiki/Graceful-Restart>.

Cautions for Graceful Restart

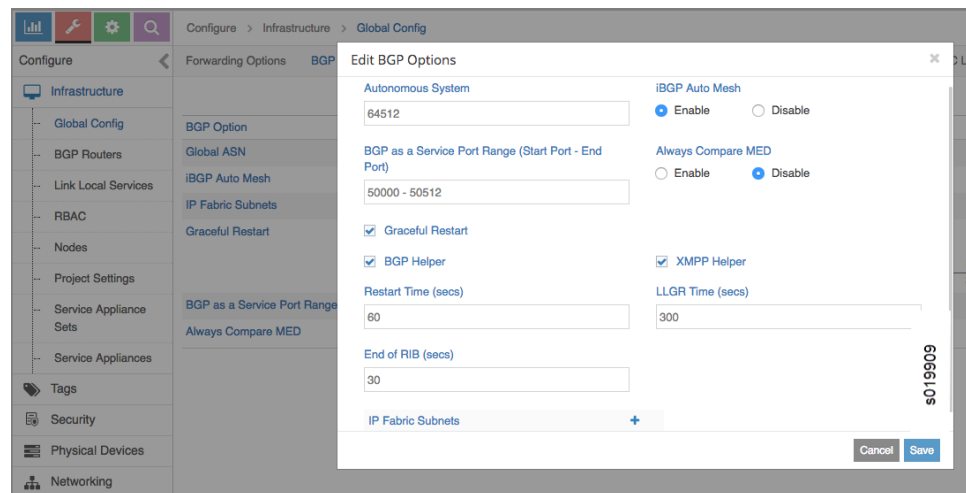
Be aware of the following caveats when configuring and using graceful restart.

- Using the graceful restart/long-lived graceful restart feature with a peer is effective either to all negotiated address families or to none. If a peer signals support for graceful restart/long-lived graceful restart for only a subset of the negotiated address families, the graceful restart helper mode does not come into effect for any family in the set of negotiated address families.
- Because graceful restart is not yet supported for contrail-vrouter-agent, the parameter should *not* be set for **graceful_restart_xmpp_helper_enable**. If the vrouter agent restarts, the data plane is reset and the routes and flows are reprogrammed anew, which typically results in traffic loss for several seconds for new and /existing flows.
- Graceful restart/long-lived graceful restart is not supported for multicast routes.
- Graceful restart/long-lived graceful restart helper mode may not work correctly for EVPN routes, if the restarting node does not preserve forwarding state for EVPN routes.

Configuring Graceful Restart with the Contrail User Interface

To configure graceful restart in the Contrail UI, go to **Configure > Infrastructure > Global Config**, then select the **BGP Options** tab. The **Edit BGP Options** window opens. Click the box for **Graceful Restart** to enable graceful restart, and enter a non-zero value for the **Restart Time**. Click the helper boxes as needed for BGP Helper and XMPP Helper. You can also enter values for the long-lived graceful restart time in seconds, and for the end of RIB in seconds. See [Figure 19 on page 78](#).

Figure 19: Configuring Graceful Restart



Related
Documentation

CHAPTER 5

Using Contrail with Kubernetes

- [Contrail Integration with Kubernetes on page 79](#)
- [Installing and Provisioning Containerized Contrail Controller for Kubernetes on page 85](#)
- [Viewing Configuration for CNI for Kubernetes on page 93](#)

Contrail Integration with Kubernetes

Contrail Release 4.0 supports the Container Network Interface (CNI) for integrating Contrail with the Kubernetes automation platform.

- [What is Kubernetes? on page 79](#)
- [Configuration Modes for Contrail Integrated with Kubernetes on page 80](#)
- [Kubernetes Services on page 82](#)
- [Ingress on page 83](#)
- [Contrail Kubernetes Solution on page 83](#)

What is Kubernetes?

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

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

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

[Table 5 on page 80](#) lists the mapping between Kubernetes concepts and OpenContrail resources.

Table 5: Kubernetes to OpenContrail Mapping

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

What is a Kubernetes Pod?

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

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

Configuration Modes for Contrail Integrated with Kubernetes

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

- [Default Mode on page 80](#)
- [Namespace Isolation Mode on page 81](#)
- [Custom Isolation Mode on page 82](#)
- [Nested Mode on page 82](#)

Default Mode

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

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



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

Namespace Isolation Mode

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

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

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

opencontrail.org/isolation : true

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

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

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

opencontrail.org/isolation.service : false

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

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

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

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

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

Custom Isolation Mode

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

"opencontrail.org/network: <fq_network_name>"

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



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

Nested Mode

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

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

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

Kubernetes Services

A Kubernetes service is an abstraction that defines a logical set of pods and the policy used to access the pods. The set of pods implementing a service are selected based on

the **LabelSelector** field in the service definition. In OpenContrail, Kubernetes service is implemented as an ECMP-native load-balancer.

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

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

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

```
kind: Service
apiVersion: v1
metadata:
  name: my-service
spec:
  selector:
    app: MyApp
  ports:
    - name: http
      protocol: TCP
      port: 80
      targetPort: 9376
    - name: https
      protocol: TCP
      port: 443
      targetPort: 9377
```

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

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

Ingress

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

Contrail Kubernetes Solution

Contrail Kubernetes solution includes the following elements.

- [Contrail Kubernetes Manager on page 84](#)
- [ECMP Load-Balancers for Kubernetes Services on page 84](#)

- [HAProxy Loadbalancer for Kubernetes Ingress on page 84](#)
- [Security Groups for Kubernetes Network Policy on page 84](#)
- [Domain Name Server \(DNS\) on page 85](#)

Contrail Kubernetes Manager

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

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

ECMP Load-Balancers for Kubernetes Services

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

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

HAProxy Loadbalancer for Kubernetes Ingress

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

See [“Using Load Balancers in Contrail” on page 453](#) for more information on load balancers.

Security Groups for Kubernetes Network Policy

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

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

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

Domain Name Server (DNS)

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

Related Documentation

- [Installing and Provisioning Containerized Contrail Controller for Kubernetes on page 85](#)
- [Viewing Configuration for CNI for Kubernetes on page 93](#)

Installing and Provisioning Containerized Contrail Controller for Kubernetes

This section describes the steps required to install and provision containerized Contrail Controller for Kubernetes.

Ensure the following prerequisites are met for successful provisioning of a Contrail Kubernetes cluster.

- An installed and running Kubernetes cluster is available.
You can choose the installation method for Kubernetes.

- Kubernetes cluster must have at least one worker node.

The Kubernetes cluster consists of one master node and at least one worker node. Kubernetes “tainted” master, a mode in which worker pods are scheduled on Kubernetes master node, is not supported.

- Ensure that Kubelet running on the Kubernetes master node does *not* have network plugin options.

If Kubelet is configured with a network plugin option:

- a. Disable or comment out the KUBELET_NETWORK_ARGS option in the configuration file.

```
/etc/systemd/system/kubelet.service.d/10-kubeadm.conf
```

- b. Restart the Kubelet service.

```
systemctl daemon-reload; systemctl restart kubelet.service
```

- Get a service account token that has a **cluster-admin** cluster role.

You can configure this token in **contrail-ansible** during provisioning of the Contrail Kubernetes cluster.

For more information, refer to the **kubernetes_access_token** variable in the **all.yml** in **contrail-ansible**.

- a. Create a service account and bind it to the **cluster-admin** cluster role.

```
kubectl create clusterrolebinding <role-binding-name> --clusterrole=cluster-admin
--serviceaccount=<service-account-name>
```

Alternatively, you can bind the **cluster-admin** role to an existing service account.

Example: Bind a **cluster-admin** role to a service account named **default**.

```
kubectl create clusterrolebinding contrail-kube-manager --clusterrole=cluster-admin
--serviceaccount=default:default
```

- b. Get the secret associated with the service account.

```
kubectl describe sa <service-account-name>
```

Example:

```
> kubectl describe sa default
Name: default
Namespace: default
Labels: <none>
Annotations: <none>
Tokens:          default-token-r353k  <-----
Image pull secrets: <none>
Mountable secrets: default-token-r353k
```

- c. Get the token associated with the secret.

```
kubectl describe secret <name>
```

Example:

```
> kubectl describe secret default-token-r353k
Name: default-token-r353k
Namespace: default
Labels: <none>
Annotations: kubernetes.io/service-account.name=default

kubernetes.io/service-account.uid=4fbcc5cf-3fed-11e7-acf4-0271c93f63d6
Type: kubernetes.io/service-account-token
Data
====
ca.crt: 1025 bytes
namespace: 7 bytes
token: $123ABC
```

Installing and Provisioning Containerized Contrail Controller for Kubernetes

Perform the following steps to install containerized Contrail controller for Kubernetes:

1. Set up password-free access to all hosts from Ansible host.

```
ssh-keygen -t rsa
ssh <user>@<host-ip> mkdir -p .ssh
ssh <user>@<host-ip> chmod 700 .ssh
cat .ssh/id_rsa.pub | ssh <user>@<host-ip> 'cat >> .ssh/authorized_keys'
```

Ensure `ssh <user>@<host-ip>` works fine.

2. Install Ansible on your Mac OS X or any other machine. Version must be = 2.2.0.

```
sudo easy_install pip
sudo pip install ansible==2.2.0
```

3. Download **contrail-kubernetes-docker_<release>_<os-name>.tgz** package and extract it. The extracted package contains **contrail-networking-tools_<release>.tgz** and **contrail-kubernetes-docker-images_<release>.tgz** packages.

The **contrail-networking-tools_<release>.tgz** contains the **contrail-ansible** package while the **contrail-kubernetes-docker-images_<release>.tgz** contains Contrail container images.

4. Extract the **contrail-ansible** package from the **contrail-networking-tools_<release>.tgz** package.

contrail-ansible is used to provision a Contrail Kubernetes cluster. The **contrail-ansible** repo contains a `site.yml` playbook that has the requisite roles and tasks to provision a fully-functional Contrail Kubernetes cluster. The inventory files in the repo expose all the parameters required by the playbook to provision the cluster. The **contrail-ansible** directory-based inventory file mechanism is recommended for provisioning.



NOTE: The scope of **contrail-ansible** is to provision only the Contrail part of the Kubernetes solution. The Kubernetes cluster should be provisioned independently using recommended Kubernetes guidelines.

Contrail Kubernetes clusters can be provisioned in the following modes:

- **Stand-alone Contrail Kubernetes cluster**

In this mode, Contrail provides networking to a stand-alone Kubernetes cluster. Contrail components are provisioned and dedicated to the management of this cluster.

- **Nested Contrail Kubernetes cluster**

In this mode, Contrail provides networking for a Kubernetes cluster that is provisioned on a Contrail OpenStack cluster. Contrail components are shared between the two clusters. Ansible provisions only the Contrail components that directly interface with the Kubernetes API server. All other Contrail components are shared between OpenStack and Kubernetes clusters.

5. Create a folder called **container_images** inside **contrail-ansible/playbook**. Copy container images to this folder by extracting **contrail-kubernetes-docker-images_<release>.tgz**.
6. Update the inventory file.

The inventory files in directory-based provisioning are as following:

- **inventory/my-inventory/hosts**. See [“inventory/my-inventory/hosts Inventory File” on page 88](#) for more information.

- **inventory/my-inventory/group_vars/all.yml**. See [“inventory/my-inventory/group_vars/all.yml Inventory File” on page 89](#) for more information.

7. Run the Ansible playbook from **contrail-ansible/playbook**.

```
ansible-playbook -i inventory/my-inventory site.yml
```

inventory/my-inventory/hosts Inventory File

This section describes the parameters and provides examples of the **inventory/my-inventory/hosts** inventory file in stand-alone and nested Contrail Kubernetes clusters.

[Table 6 on page 88](#) lists the parameters used in the **inventory/my-inventory/hosts** inventory file.

In [Table 6 on page 88](#), **Cluster Mode** is one of the following:

- Stand-alone —Applicable only to a stand-alone cluster.
- Nested —Applicable only to nested cluster.
- Both—Applicable to both stand-alone and nested clusters.

Table 6: Parameters in inventory/my-inventory/hosts

Parameter	Cluster Mode	Description
contrail-repo	Nested	List of hosts where contrail apt or yum repo container will be started. This repo will be used by other nodes on installing any packages in the node. Setting up contrail-cni needs this repo enabled
contrail-controllers	Stand-alone	List of hosts where contrail-controller container or processes are to be provisioned. .
contrail-analyticsdb	Stand-alone	List of hosts where contrail-analyticsdb container or process is to be provisioned.
contrail-analytics	Stand-alone	List of hosts where contrail-analytics container or process is to be provisioned.
contrail-kubernetes	Both	Node where contrail-kube-manager container or process is to be run.
contrail-compute	Both	List of hosts which are to be provisioned as kubernetes compute/minion nodes. Contrail vRouter or vrouter-agent or CNI will be provisioned on these nodes.
kubernetes-contrail-controllers	Nested	List of nodes with pre-existing contrail-controller container or processes to which contrail-kube-manager should connect to.
kubernetes-contrail-analytics	Nested	List of nodes with pre-existing contrail-analytics container or processes to which contrail-kube-manager should connect to.

Example: inventory/my-inventory/hosts File in a Stand-alone Contrail Kubernetes Cluster

The following is an example of the **inventory/my-inventory/hosts** file in a stand-alone Contrail Kubernetes cluster:

```
[contrail-controllers]
10.xx.27.16

[contrail-analyticsdb]
10.xx.27.16

[contrail-analytics]
10.xx.27.16

[contrail-kubernetes]
10.xx.27.16

[contrail-compute]
10.xx.23.37
```

Example: Nested inventory/my-inventory/hosts File in a Nested Contrail Kubernetes Cluster

The following is an example of the **inventory/my-inventory/hosts** file in a nested Contrail Kubernetes cluster:

```
[contrail-repo]
10.xx.31.71

[contrail-kubernetes]
10.xx.31.71

[contrail-compute]
10.xx.31.72

[kubernetes-contrail-controllers]
10.xx.29.27

[kubernetes-contrail-analytics]
10.xx.29.27
```

inventory/my-inventory/group_vars/all.yml Inventory File

This section describes the parameters and provides examples of the **inventory/my-inventory/group_vars/all.yml** inventory file in stand-alone and nested Contrail Kubernetes clusters.

[Table 7 on page 90](#) describes the configuration parameters used in the **inventory/my-inventory/group_vars/all.yml** inventory file.

In [Table 7 on page 90](#), **Cluster Mode** is one of the following:

- Stand-alone —Applicable only to a stand-alone cluster.
- Nested —Applicable only to nested cluster.
- Both—Applicable to both stand-alone and nested clusters.

Table 7: Parameters in inventory/my-inventory/group_vars/all.yml

Parameter	Value	Default	Cluster Mode	Description
cloud_orchestrator	Kubernetes	None	Both	Specifies orchestrator type.
contrail_compute_mode	container	bare_metal	Both	Specifies if the Contrail components must be run as containers or as processes on a stand-alone server.
keystone_config	{ip: <ip>, admin_password: <passwd>, admin_user: <username>, admin_tenant: <tenant-name>}	None	Nested	Keystone authentication information.
nested_cluster_private_network	"<cluster-private-CIDR>"	None	Nested	The IP subnet reserved for use by Kubernetes for internal cluster management and housekeeping. The Ansible user is responsible to make sure this CIDR does not collide with existing CIDRs in the virtual-network.
kubernetes_cluster_name	<cluster-name>	k8s-default	Both	Name of the Kubernetes cluster being provisioned.
nested_cluster_network	{domain: <name>, project: <name>, name: <name>}	None	Nested	Virtual Network in which the Kubernetes cluster must be provisioned. This network must be the same network to which the virtual machines that host the Kubernetes cluster belong.
kubernetes_access_token	< token >	None	Both	RBAC token to connect to Kubernetes API server.
nested_mode	true	None	Nested	Parameter to enable nested provisioning of a Kubernetes cluster.
kubernetes_public_fip_pool	{domain: <id>, project: <id>, network: <id>, name: <id>}	None	Both	Kubernetes FloatingIpPool to be used for service or ingress.
kubernetes_cluster_project	{domain: <id>, project: <id>}	{domain: default-domain, project: default}	Both	Fq-name of Contrail project within which Kubernetes cluster must be provisioned.
kubernetes_pod_subnet	<CIDR>	10.32.0.0/12	Both	Pod subnet used by Kubernetes cluster.
kubernetes_service_subnet	<CIDR>	10.96.0.0/12	Both	Service subnet used by Kubernetes cluster.

```
docker_install_method: package
docker_py_pkg_install_method: pip

# ansible connection details
ansible_user: root
ansible_become: true
ansible_ssh_private_key_file: ~/.ssh/id_rsa

contrail_compute_mode: container

os_release: ubuntu14.04

# contrail version
contrail_version: 4.0.0.0-16

cloud_orchestrator: kubernetes

# vrouter physical interface
vrouter_physical_interface: enp6s0f0

# global_config:

analytics_api_config: {aaa_mode: no-auth}

# To configure custom webui http port
webui_config: {http_listen_port: 8085}

# Name of the kubernetes cluster being provisioned.
kubernetes_cluster_name: k8s5

# Access token to connect to Kuberenetes API server.
kubernetes_access_token: eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJrdWJlcm5ldGVzL3N1cnZpY2VhY2NvdW50Iiwia3ViZXJuZXRlcy5pby9zZXJ2YWNIYWVjb3VudC9uYW1lc3BhY2UiOiJkZWZhdxw0Iiwia3ViZXJuZXRlcy5pby9zZXJ2YWNIYWVjb3VudC9zZWNYZXQubmFtZSI6ImRlZmF1bHQtdG9rZW4tcTUzYmYiLCJrdWJlcm5ldGVzMlrlL3N1cnZpY2VhY2NvdW50L3N1cnZpY2UtYWNjb3VudC5uYW1lc3BhY2UiOiJjoiZGVmYXVsdCIsImt1YmVybmV0ZXMuaW8vc2Vydm1lZmFjY291bnQvc2VydmljZS1hY2NvdW50LnVpZCI6ImVhNzE1YjJkLTJhYWUtMTFlNy1iZmJmLTAyMWQwOTNhMzRkMSIsInN1YiI6InN5c3R1bTpzX2J2aWNIYWVjb3VudDpkZWZhdWx0OmRlZmF1bHQifQ.Kj0-NYBopRc8rMsX4nnKpJa570k2iamPOgCTdj3d93MW20girt4IgdAmR4v4kiFQ0-h5eYGVlFA3ftkPuwB5GbHDZ9x7BoYc7b759i2cuX3AMtbC15kNcbGY7_7JPIDkMHwwRj7FK7Y57eFTstCxcpr4itqxzsRi7jcOnrrrcBDkv1OkDhA93ID4ChPwE2PcsAf_LV9ds-gSzuyPIQt0qdxnQvI262AJgeNowbQhkYguoqZWJIE--AwpgSEONiNpjciUx1HC2uarSP3q9mMr2q4YOHRjxJwuz3fukaSRNZyQEpyE5GSWKXTefc
```

```
7h52R5Kphn2nT9gg6x175mrrnNQ
# Kubernetes API server IP.
kubernetes_api_server: 10.14.27.16
```

Example: inventory/my-inventory/group_vars/all.yml File in a Nested Contrail Kubernetes Cluster

The following is an example of the `inventory/my-inventory/group_vars/all.yml` file in a nested Contrail Kubernetes cluster

[illegible]

```
ydm1jZWFjY291bnQvc2VydmljZS1hY2NvdW50LnVpZCI6ImVhNzE1YjJkLTJhY
WUtMTF1Ny1iZmJmLTAYMWQwOTNhMzRkMSIsInN1YiI6InN5c3R1bTpzZXJ2
aWN1YWYjY291bnQvc2VydmljZS1hY2NvdW50LnVpZCI6ImVhNzE1YjJkLTJhY
KpJa570k2iamPOgCTdj3d93MW20gi rt4IgdAmR4v4kifQ0-h5eYGV1fA3ftkPu
Wb5GbHDz9x7BoYc7b759i2cuX3AmtbC15kNcbGY7_7JPIDkMHwRj7FK7Y
57eEFTstCxcprR4itqxsRi7jc0nrrcbDkv10kDhA93ID4ChPwE2PcsAf_LV9ds-g
SzuyPIQt0qdxnQvI262AjgeNowbQhkYguoqZWJIE--AwpgSEONiNpjcxixUx1HC2
uaRSP3g9mMr2g4YQHRjxJwuz3fUkaSRNZyQEpyE5G5WKXTefc7h52R5Kph
n2nT9gg6x175mrrnNQ
```

```
# Kubernetes cluster is nested within an Openstack cluster.
nested_mode: true
```

```
# Kubernetes API server IP.
kubernetes_api_server: 10.14.27.16
```

- Related Documentation**
- [Contrail Integration with Kubernetes on page 79](#)
 - [Viewing Configuration for CNI for Kubernetes on page 93](#)

Viewing Configuration for CNI for Kubernetes

Use the verification steps in this topic to view and verify your configuration of Contrail Container Network Interface (CNI) for Kubernetes.

- [View Pod Name and IP Address on page 93](#)
- [Verify Reachability of Pods on page 93](#)
- [Verify If Isolated Namespace-Pods Are Not Reachable on page 94](#)
- [Verify If Non-Isolated Namespace-Pods Are Reachable on page 94](#)
- [Verify If a Namespace is Isolated on page 95](#)

View Pod Name and IP Address

Use the following command to view the IP address allocated to a pod.

```
[root@device ~]# kubectl get pods --all-namespaces -o wide
```

NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE	IP
	NODE					
default	client-1	1/1	Running	0		19d
	10.47.25.247 k8s-minion-1-3					
default	client-2	1/1	Running	0		19d
	10.47.25.246 k8s-minion-1-1					
default	client-x	1/1	Running	0		19d
	10.84.21.272 k8s-minion-1-1					

Verify Reachability of Pods

Perform the following steps to verify if the pods are reachable to each other.

1. Determine the IP address and name of the pod.

```
[root@device ~]# kubectl get pods --all-namespaces -o wide
```

NAME	READY	STATUS	RESTARTS	AGE	IP
example1-36xpr	1/1	Running	0	43s	10.47.25.251
b3s37					

```
example2-pldp1 1/1 Running 0 39s 10.47.25.250
b3s37
```

2. Ping the destination pod from the source pod to verify if the pod is reachable.

```
root@device ~]# kubectl exec -it example1-36xpr ping 10.47.25.250
PING 10.47.25.250 (10.47.25.250): 56 data bytes
64 bytes from 10.47.25.250: icmp_seq=0 ttl=63 time=1.510 ms
64 bytes from 10.47.25.250: icmp_seq=1 ttl=63 time=0.094 ms
```

Verify If Isolated Namespace-Pods Are Not Reachable

Perform the following steps to verify if pods in isolated namespaces cannot be reached by pods in non-isolated namespaces.

1. Determine the IP address and name of a pod in an isolated namespace.

```
[root@device ~]# kubectl get pod -n test-isolated-ns -o wide
NAME                                READY   STATUS    RESTARTS   AGE      IP
example3-bvqx5 1/1      Running   0          1h       10.47.25.249
b3s37
```

2. Determine the IP address of a pod in a non-isolated namespace.

```
[root@device ~]# kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
example1-36xpr 1/1      Running   0          15h
example2-pldp1 1/1      Running   0          15h
```

3. Ping the IP address of the pod in the isolated namespace from the pod in the non-isolated namespace.

```
[root@device ~]# kubectl exec -it example1-36xpr ping 10.47.25.249
--- 10.47.255.249 ping statistics ---
2 packets transmitted, 0 packets received, 100% packet loss
```

Verify If Non-Isolated Namespace-Pods Are Reachable

Perform the following steps to verify if pods in non-isolated namespaces can be reached by pods in isolated namespaces.

1. Determine the IP address of a pod in a non-isolated namespace.

```
[root@device ~]# kubectl get pods -o wide
NAME                                READY   STATUS    RESTARTS   AGE      IP
example1-36xpr 1/1      Running   0          15h       10.47.25.251
b3s37
example2-pldp1 1/1      Running   0          15h       10.47.25.250
b3s37
```

2. Determine the IP address and name of a pod in an isolated namespace.

```
[root@device ~]# kubectl get pod -n test-isolated-ns -o wide
NAME                                READY   STATUS    RESTARTS   AGE      IP
```

	NODE				
example3-bvqx5 b3s37	1/1	Running	0	1h	10.47.25.249

3. Ping the IP address of the pod in the non-isolated namespace from a pod in the isolated namespace.

```
[root@device ~]# kubectl exec -it example3-bvqx5 -n test-isolated-ns ping 10.47.25.251
PING 10.47.25.251 (10.47.25.251): 56 data bytes
64 bytes from 10.47.25.251: icmp_seq=0 ttl=63 time=1.467 ms
64 bytes from 10.47.25.251: icmp_seq=1 ttl=63 time=0.137 ms
^C--- 10.47.25.251 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.137/0.802/1.467/0.665 ms
```

Verify If a Namespace is Isolated

Namespace annotations are used to turn on isolation in a Kubernetes namespace. In isolated Kubernetes namespaces, the namespace metadata is annotated with the `opencontrail.org/isolation: true` annotation.

Use the following command to view annotations on a namespace.

```
[root@a7s16 ~]# kubectl describe namespace test-isolated-ns
Name:          test-isolated-ns
Labels:        <none>
Annotations:   opencontrail.org/isolation : true      Namespace is isolated
Status:        Active
```

- Related Documentation**
- [Contrail Integration with Kubernetes on page 79](#)
 - [Installing and Provisioning Containerized Contrail Controller for Kubernetes on page 85](#)

CHAPTER 6

Using Contrail with VMware vCenter

- [Installing and Provisioning VMware vCenter with Contrail on page 97](#)
- [Underlay Network Configuration for ContrailVM on page 106](#)
- [Sample Testbed.py Files for Contrail vCenter on page 114](#)
- [Using the Contrail and VMware vCenter User Interfaces to Manage the Network on page 119](#)

Installing and Provisioning VMware vCenter with Contrail

- [Overview: Integrating Contrail with vCenter Server on page 97](#)
- [Different Modes of vCenter Integration with Contrail on page 98](#)
- [vCenter-Only Mode on page 98](#)
- [vCenter-as-Compute Mode on page 99](#)
- [Preparing the Installation Environment on page 100](#)
- [Installation for vCenter-Only Mode on page 100](#)
- [Installing the vCenter-Only Components on page 101](#)
- [Installation for vCenter-as-Compute Mode on page 102](#)
- [Installing the vCenter-as-Compute Components on page 104](#)
- [Verification on page 105](#)
- [Adding Hosts or Nodes on page 105](#)
- [Adding an ESXi Host to an Existing vCenter Cluster on page 105](#)
- [Adding a vCenter Cluster to vCenter-as-Compute on page 105](#)

Overview: Integrating Contrail with vCenter Server

This topic describes how to install and provision Contrail Release 2.20 and later so that it works with existing or already provisioned vSphere deployments that use VMware vCenter as the main orchestrator.

The Contrail VMware vCenter solution is comprised of the following main components:

- Control and management that runs the following components as needed per Contrail system:
 - A VMware vCenter Server independent installation that is not managed by Juniper Networks Contrail. The Contrail software provisions vCenter with Contrail components and creates entities required to run Contrail.
 - The Contrail controller, including the configuration nodes, control nodes, analytics, database, and Web UI, which are installed, provisioned, and managed by Contrail software.
 - A VMware vCenter plugin provided with Contrail.
- VMware ESXi virtualization platforms forming the compute cluster, with Contrail data plane (vRouter) components running inside an Ubuntu-based virtual machine. The virtual machine, named *ContrailVM*, forms the compute personality while performing Contrail installs. The ContrailVM is set up and provisioned by Contrail. There is one ContrailVM running on each ESXi host.

Different Modes of vCenter Integration with Contrail

The vCenter integrated Contrail solution has the following modes:

- vCenter-only
- vCenter-as-compute

vCenter-Only Mode

In the vCenter-only mode, vCenter is the main orchestrator, and Contrail is integrated with vCenter for the virtual networking.

[Figure 20 on page 99](#) shows the Contrail vCenter-only solution.

99



Figure 21 on page 99 shows the Contrail vCenter-as-compute solution.

The diagram illustrates the VMware vCenter and ESXi architecture. At the top, a user icon is connected to a 'Compute Orchestration' box, which in turn connects to 'vmware VCenter'. The 'vmware VCenter' is connected to a 'Contrail VM' and an 'ESXi' host. The 'Contrail VM' contains a 'Linux Internal' section with 'vRouter Agent' and 'vRouter' components. The 'vRouter' is connected to a 'FAB-PG' (Fabric Port Group) on 'VMware vSwitch0'. The 'FAB-PG' is connected to a 'Physical NIC'. The 'vRouter' is also connected to a 'Tagged (trunk)' port on 'VMware vSwitch0'. The 'Tagged (trunk)' port is connected to a 'VMware vSwitch' which has two virtual ports: 'Vlan G' and 'Vlan R'. 'Vlan G' is connected to 'VM G2' and 'Vlan R' is connected to 'VM R2' and 'VM R3'. The 'ESXi' host is connected to a 'Contrail Controller' which is connected to a 'Network Orchestration' box. The 'Network Orchestration' box is connected to a 'Contrail VM' and an 'ESXi' host. The 'ESXi' host is connected to a 'vRouter' which is connected to 'VM R1', 'VM R4', and 'VM G1'. The 'vRouter' is also connected to a 'KVM' (Kernel-based Virtual Machine) host.

Preparing the Installation Environment

Use the standard Contrail installation procedure to install Contrail on one of the target boxes or servers, so that Fabric (fab) scripts can be used to install and provision the entire cluster.

Follow the steps in the *Installing Contrail Packages for Ubuntu* section in *Installing the Contrail Packages, Part One (CentOS or Ubuntu)*



NOTE: The fab scripts require a file named `testbed.py`, that holds all of the key attributes for fab to begin provisioning, including the IP addresses of the Contrail roles. Refer to the sample `testbed.py` file for Contrail vCenter in “[Sample Testbed.py Files for Contrail vCenter](#)” on page 114.

Installation for vCenter-Only Mode

This section lists the basic installation procedure and the assumptions and prerequisites necessary before starting the installation of any VMware vCenter Contrail integration.



NOTE: To ensure that you are using the correct versions of all software for your specific system, refer to the Supported Platforms section in the release notes for your release of Contrail, such as [Release Notes for Contrail 3.0](#) and the like.

- Installation:** The following assumptions and prerequisites are required for a successful installation of a VMware vCenter Contrail integrated system:
- Assumptions and Prerequisites**
- VMware vCenter Server
 - A cluster of ESXi hosts with VMware
 - The following software installation packages:
 - The `contrail-install-packages_x.x.x.x.vcenter_all.deb` package for Ubuntu 14.04
 - VMDK image of ContrailVM
 - Because a Contrail vRouter runs as a virtual machine on each ESXi host, it needs an IP address assigned from the same underlay network as the host, all of which must be specified appropriately in the `testbed.py` file. Refer to the section [“Underlay Network Configuration for ContrailVM” on page 106](#) for ContrailVM IP fabric connectivity.

Installing the vCenter-Only Components

Follow the steps in this section to install the Contrail for vCenter-only components. Refer to the sample `testbed.py` file for Contrail vCenter for specific examples: [“Sample Testbed.py Files for Contrail vCenter” on page 114](#).

1. Ensure that all information in the `esxi_hosts` section of the `testbed.py` file is accurate, then provision the ESXi hosts using the following command:

```
fab prov_esxi
```

The `esxi_hosts = { }` section of the `testbed.py` file spawns the ContrailVM from the bundled VMDK file.

- Ensure that all required information in the section is specific to your environment and that the VMDK file can be accessed by the machine running the `fab` task.
- If the IP address and the corresponding MAC address of the ContrailVM are statically mapped in the DHCP server, specify the static IP address in the `host` field and the MAC address in the `mac` field in the `contrail_vm` subsection.
- ContrailVM IP fabric connectivity can be configured in various ways; refer to the section [“Underlay Network Configuration for ContrailVM” on page 106](#) for details.

When finished, ping each of the ContrailVMs to make sure they are reachable.

2. Set up vCenter.

```
fab setup_vcenter
```

Specify the orchestrator to be vCenter for proper provisioning of vCenter-related components, as in the following:

```
env.orchestrator = 'vcenter'
```

When finished, verify that you can see the ESXIs and Contrail VMs on the vCenter user interface, refer to [“Using the Contrail and VMWare vCenter User Interfaces to Manage the Network”](#) on page 119.

3. Ensure that the Contrail Debian package is available on all the nodes.

```
fab install_pkg_all:<Contrail deb package>
```

4. Install the Contrail components into the desired roles on the specified nodes.

```
fab install_contrail
```

This also installs the vCenter plugin on the Contrail config nodes.

5. Set up the management and control data interfaces. Perform this step ONLY if the management and control_data interfaces are separate.

```
fab setup_interface_node
```

6. Provision all of the Contrail components and the vCenter plugin.

```
fab setup_all
```

This step also creates the required configuration files on the system.

Installation for vCenter-as-Compute Mode

This section lists the basic installation procedure and the assumptions and prerequisites necessary before starting the installation of any VMware vCenter-as-compute Contrail integration.



NOTE: To ensure you are using the correct versions of all software for your specific system, refer to the Supported Platforms section in the release notes for your release of Contrail.

**Installation:
Assumptions and
Prerequisites**

The following assumptions and prerequisites are required for a successful installation of a VMware vCenter Contrail integrated system:

- VMware vCenter Server
- A cluster of ESXi hosts with VMware
- The following software installation packages:
 - The OpenStack contrail-install *.deb package for Ubuntu 14.04
 - VMDK image of ContrailVM
 - Releases up to Contrail 3.0 only: The contrail-install-vcenter-plugin *.deb package.
- Because a Contrail vRouter runs as a virtual machine on each ESXi host, it needs an IP address assigned from the same underlay network as the host, all of which must be specified appropriately in the **testbed.py** file. Refer to [“Underlay Network Configuration for ContrailVM” on page 106](#) for ContrailVM IP fabric connectivity.

For the vCenter-as-compute mode, an additional role of **'vcenter-compute'** is required, specified as **['vcenter_compute']** in the **env.roledefs** section of the **testbed.py** file. Nodes configured as **vcenter_compute** act as the **nova-compute** nodes in this mode.

For specific examples, refer to the sample **testbed.py** file in [“Sample Testbed.py Files for Contrail vCenter” on page 114](#).

Installing the vCenter-as-Compute Components

To install the vCenter-as-compute components:

1. Provision the ESXi hosts.

fab prov_esxi

Before performing this, ensure that all information in the **esxi_hosts** section of the **testbed.py** file is accurate.

The **esxi_hosts = { }** section of the **testbed.py** file spawns the ContrailVM from the bundled VMDK file.

Ensure all required information in the section is specific to your environment and the VMDK file can be accessed by the machine running the fab task.

If the IP address and the corresponding MAC address of the ContrailVM are statically mapped in the DHCP server, specify the static IP address in the **host** field and the MAC address in the **mac** field in the **contrail_vm** subsection.

ContrailVM IP fabric connectivity can be configured in various ways, refer to [“Underlay Network Configuration for ContrailVM” on page 106](#) for details.

When finished, ping each of the ContrailVMs to make sure they are reachable.

2. Set up vCenter.

fab setup_vcenter

Specify the orchestrator to be OpenStack for proper provisioning of vCenter-as-compute, as in the following:

```
env.orchestrator = 'openstack'
```

In the vCenter-as-compute mode, you can have multiple **vcenter_servers** specified in the **testbed.py**.

Refer to [“Sample Testbed.py Files for Contrail vCenter” on page 114](#).

When finished, verify that you can see the ESXIs and ContrailVMs on the vCenter user interface. Refer to [“Using the Contrail and VMWare vCenter User Interfaces to Manage the Network” on page 119](#)

3. Ensure that the contrail-install.deb package is available on all nodes.

```
fab install_pkg_all: <Contrail deb package>
```

4. Install the Contrail components into the desired roles on the specified nodes.

```
fab install_contrail
```

5. Set up the management and control data interfaces. Run this step ONLY if the management and control_data interfaces are separate.


```
fab setup_interface_node
```

6. Provision all of the Contrail components and the vCenter plugin.

```
fab setup_all
```

This step also creates the required configuration files on the system.

Verification

When the provisioning step completes, run the **contrail-status** command on all nodes to view a health check of the Contrail configuration and control components.

Adding Hosts or Nodes

You can add some vCenter features to existing installations, including:

- Adding an ESXi host
- Adding a vCenter cluster

Adding an ESXi Host to an Existing vCenter Cluster

You can provision and add an ESXi host to an existing vCenter cluster.

To add an ESXi host, use the following commands (which spawn the compute VM on an ESXi host) install and set up Contrail roles, and add the ESXi host to the vCenter cluster and switch:

1. Spawn the Contrail VM.

```
fab prov_esxi: <esxi_host>
```

where *<esxi_host>* is the ESXi hostname as specified in the **esxi_hosts{}**.

2. Install the Contrail *.deb package on the Contrail VM in the ESXi host.

```
fab install_pkg_node:<contrail-deb>,root@ContrailVM-ip
```

3. Add the ESXi to the vCenter cluster and put it into the switch, as specified in the configuration in the **testbed.py** file.

```
fab add_esxi_to_vcenter:esxi_host
```

4. Install and set up the Contrail vRouter on the Contrail VM in the ESXi host.

```
fab add_vrouter_node:root@ContrailVM-ip
```

Adding a vCenter Cluster to vCenter-as-Compute

Use this procedure to add a vCenter cluster to a vCenter-as-compute system. Ensure that you have provisioned and added all of the ESXi hosts, as described in the previous *Adding an ESXi Host to an Existing vCenter Cluster* procedure.

To set up and add a vCenter compute node:

1. Install the Contrail *.deb package on the vCenter compute nodes.

```
fab install_pkg_node:<contrail-deb>,root@<vcenter_compute-ip>
```

2. **For releases up to Contrail 3.0 only:** Install the vcenter-plugin in the vcenter_compute node.

```
fab install_contrail_vcenter_plugin:<vcenter-plugin-deb>,root@<vcenter_compute-ip>
```

3. Provision the vcenter_compute node and set up Nova configuration files.

```
fab add_vcenter_compute_node:root@vcenter_compute-ip
```

Related Documentation

- [Underlay Network Configuration for ContrailVM on page 106](#)
- [Sample Testbed.py Files for Contrail vCenter on page 114](#)
- [Using the Contrail and VMWare vCenter User Interfaces to Manage the Network on page 119](#)

Underlay Network Configuration for ContrailVM

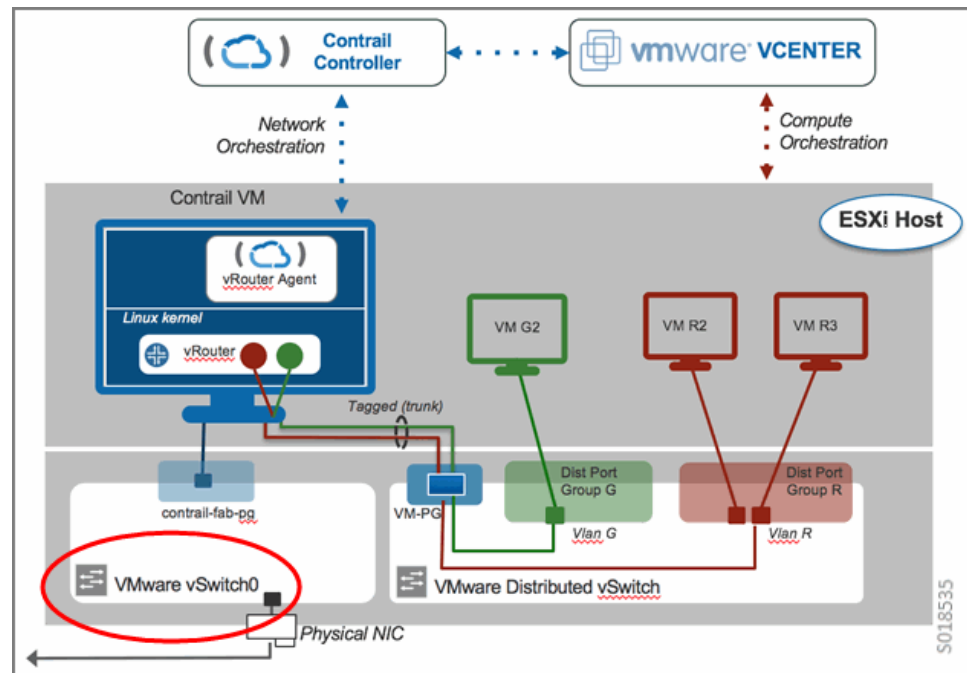
When using vCenter as compute, the ContrailVM can be configured in several different ways for the underlay (**ip-fabric**) connectivity:

- [Standard Switch Setup on page 106](#)
- [Distributed Switch Setup on page 107](#)
- [PCI Pass-Through Setup on page 109](#)
- [SR-IOV Setup on page 110](#)

Standard Switch Setup

In the standard switch setup, the ContrailVM is provided an interface through the standard switch port group that is used for management and control data, see [Figure 22 on page 107](#).

Figure 22: Standard Switch Setup



To set up the ContrailVM in this mode, the standard switch and port group must be configured in the **contrail_vm** section in the server JSON configuration.

If not configured, the default value of **vSwitch0** is used for the standard switch.

The ContrailVM supports multiple NICs for management and control_data interfaces. The management interface must have DHCP flag as true and the control_data interface can have DHCP set as false. When DHCP is set to false the interface script is updated with the IP address as specified in the server JSON file. Additional configuration, like static routes and bond interface, can be configured in the server JSON in Server Manager.

The following is an example of server configuration in Server Manager with standard switch.

```
"contrail_vm": {
    "mgmt_switch": "vSwitch0",
    "mgmt_pg": "mgmt-pg"
    "control_data_switch": "vSwitch1",
    "control_data_pg": "control-pg",
    "vmdk": "/root/vmdk_new/vmdk.tar"
}
```

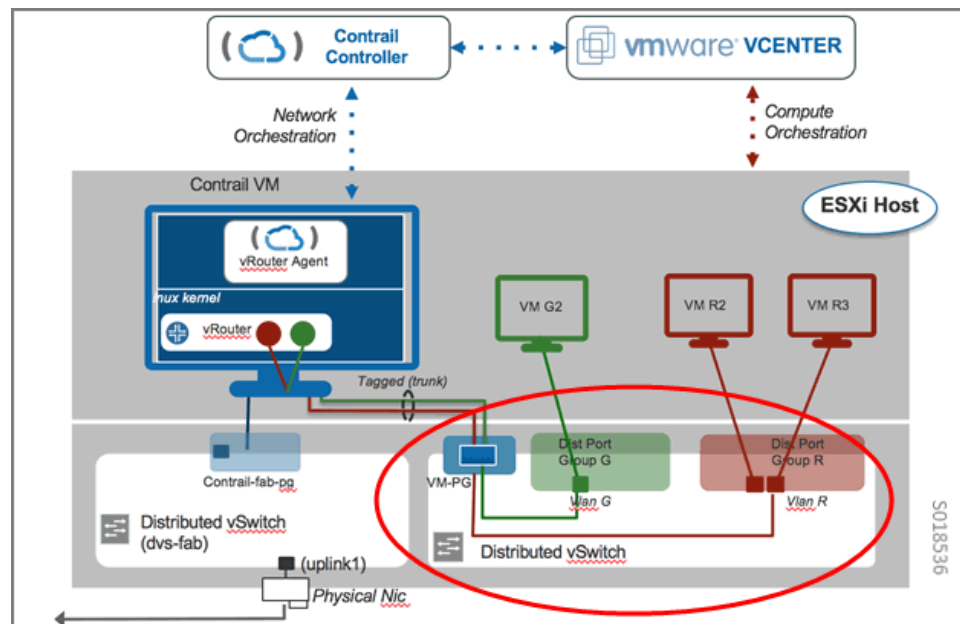
Distributed Switch Setup

A distributed switch functions as a single virtual switch across associated hosts.

In the distributed switch setup, the ContrailVM is provided an interface through the distributed switch port group that is used for management and control data, see [Figure 23 on page 108](#).

The ContrailVM can be setup to use the management and **control_data** NICs from DVS. The DVS configuration for **control_data** and management is provided in cluster JSON configuration in Server Manager. When the DVS configuration is specified then the standard switch configuration is ignored.

Figure 23: Distributed Switch Setup



To set up the Contrail VM in this mode, configure the distributed switch, port group, number of ports in the port group, and the uplink in the **vcenter_servers** section in cluster JSON configuration.



NOTE: The uplink can be a link aggregation group (LAG). If you use LAG, then DVS and LAG need to be pre-configured.

The following is an example distributed switch configuration in the cluster JSON file.

```
"vcenter_servers": [
    {
        "server1": {
            "datacenters": {
                "i27_datacenter11": {
                    "dv_switch_control_data": {
                        "dv_port_group_control_data": {
                            "dv_portgroup_name": "pg_name",

                            "number_of_ports": "3",
                            "uplink": "vmnic1"
                        },
                        "dv_switch_name": "dvs_name"
                    },
                    "dv_switch_mgmt": {
                        "dv_port_group_mgmt": {
```

```

        "dv_portgroup_name": "",
        "number_of_ports": "",
        "uplink": ""
    },
    "dv_switch_name": ""
}
}
}
}
}
]

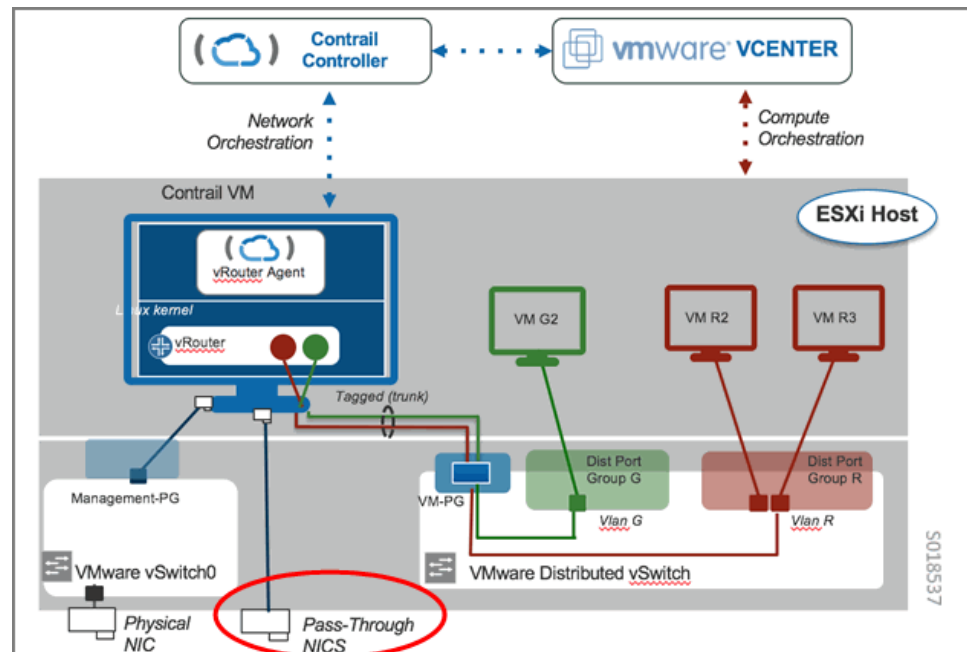
```

PCI Pass-Through Setup

PCI pass-through is a virtualization technique in which a physical Peripheral Component Interconnect (PCI) device is directly connected to a virtual machine, bypassing the hypervisor. Drivers in the VM can directly access the PCI device, resulting in a high rate of data transfer.

In the pass-through setup, the ContrailVM is provided management and control data interfaces. Pass-through interfaces are used for control data. [Figure 24 on page 109](#) shows a PCI pass-through setup with a single **control_data** interface.

Figure 24: PCI Pass-Through with Single Control Data Interface



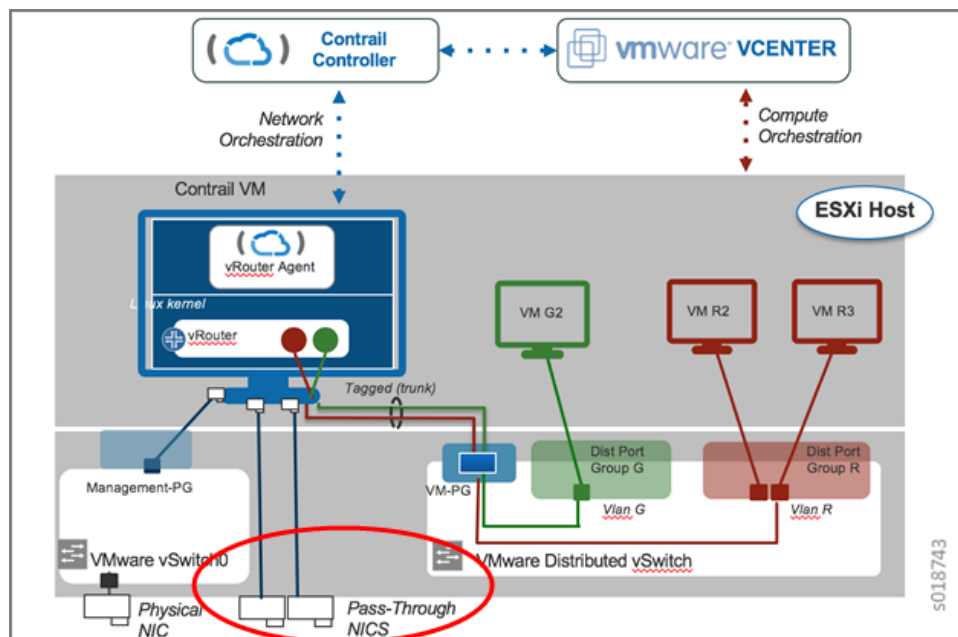
When setting up the ContrailVM with pass-through interfaces, upon provisioning ESXi hosts in the installation process, the PCI pass-through interfaces are exposed as Ethernet interfaces in the ContrailVM, and are identified in the **control_data** device field.

The following is an example PCI pass-through configuration with a single **control_data** interface:

```
'contrail_vm': {
    66 55 66 55 66 55
    "pci_devices": {
        "nics": ["04:00.0"],
    },
    "vmdk": "/root/vmdk_new/vmdk.tar"
}
```

Figure 25 on page 110 shows a PCI pass-through setup with a bond_control data interface, which has multiple pass-through NICs.

Figure 25: PCI Pass-Through Setup with Bond Control Interface



Update the ContrailVM section in server JSON configuration with `pci_devices` as shown in the following example. Refer to the Server Manager documentation for bond interface-configuration in server JSON configuration.

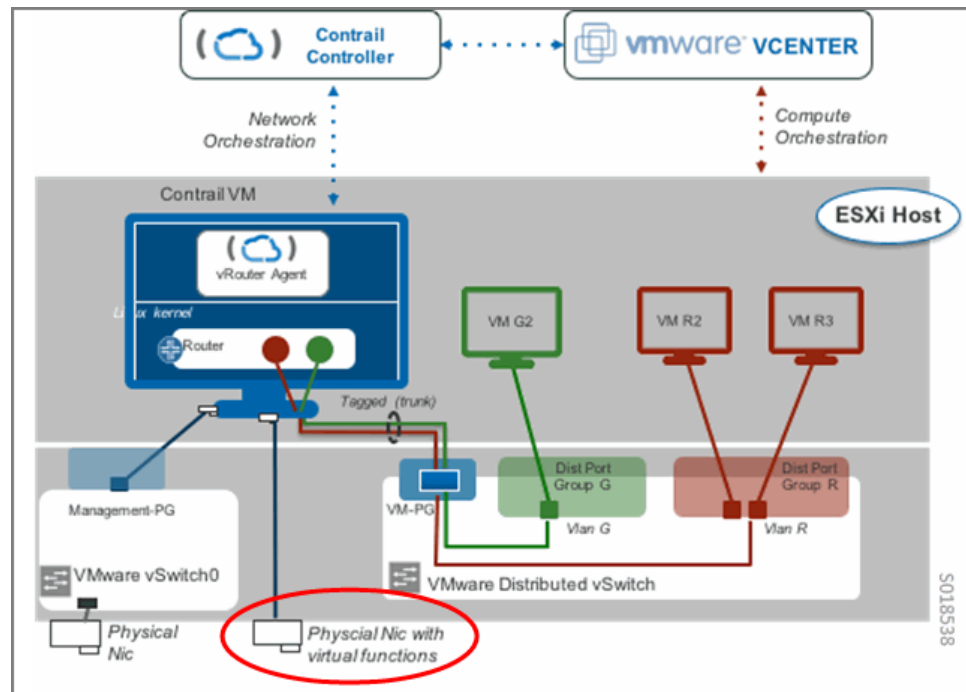
```
"contrail_vm": {
    "pci_devices": {
        "nics": ["04:00.0", "04:00.1"]
    }
    "vmdk": "/root/vmdk_new/vmdk.tar"
}
```

SR-IOV Setup

A single root I/O virtualization (SR-IOV) interface allows a network adapter device to separate access to its resources among various hardware functions.

In the SR-IOV setup, the ContrailVM is provided management and control data interfaces. SR-IOV interfaces are used for control data. See [Figure 26 on page 111](#).

Figure 26: SR-IOV Setup



In VMware, the **port-group** is mandatory for SR-IOV interfaces because the ability to configure the networks is based on the active policies for the port holding the virtual machines. For more information, refer to VMware's [SR-IOV Component Architecture and Interaction](#).

The **port-group** is created as part of provisioning; however, before the provisioning, the distributed virtual switch (DVS) for the **port-group** should be created by the user.

To set up the ContrailVM with SR-IOV interfaces, all **JSON** file configurations used for the standard switch setup are also used for the pass-through setup, providing the management connectivity to the ContrailVM.

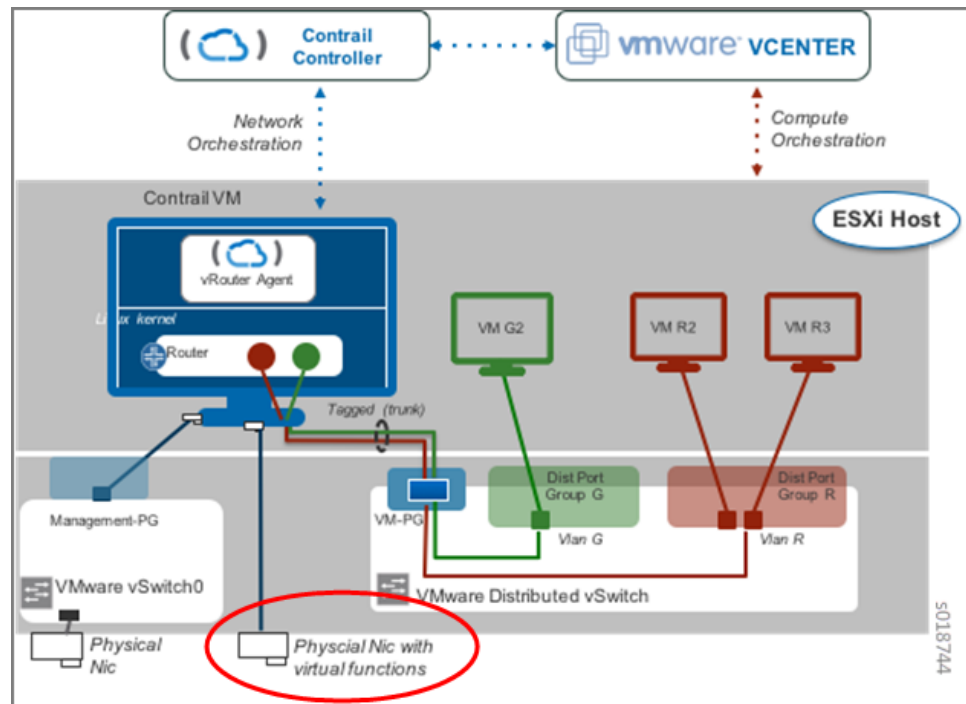
To provide the **control_data** interfaces, configure the SR-IOV-enabled physical interfaces in the **contrail_vm** section, and configure the **control_data** in the global section of **testbed.py**.

Configure the port group (**dv_port_group_sr_iov**) and the DVS (**dv_switch_sr_iov**) in the cluster JSON configuration in Server Manager.

Upon provisioning ESXi hosts in the installation process, the SR-IOV interfaces are exposed as Ethernet interfaces in the ContrailVM.

Figure 27 on page 112 shows a SR-IOV setup with a single **control_data** interface.

Figure 27: SR-IOV With Single Control Data Interface



The following are examples of SR-IOV configuration for the cluster and server configuration.

The cluster configuration:

```
"vcenter_servers": [
  {
    "server1": {
      "datacenters": {
        "i27_datacenter11": {
          "dv_switch_sr_iov": {
            "dv_port_group_sr_iov": {
              "dv_portgroup_name": "",
              "number_of_ports": "",
              "uplink": ""
            },
            "dv_switch_name": ""
          }
        }
      }
    }
  }
]
```

The server configuration:

```
"contrail_vm": {
  "sr_iov_nics": {
    "nics": ["vmnic0"]
  }
}
```



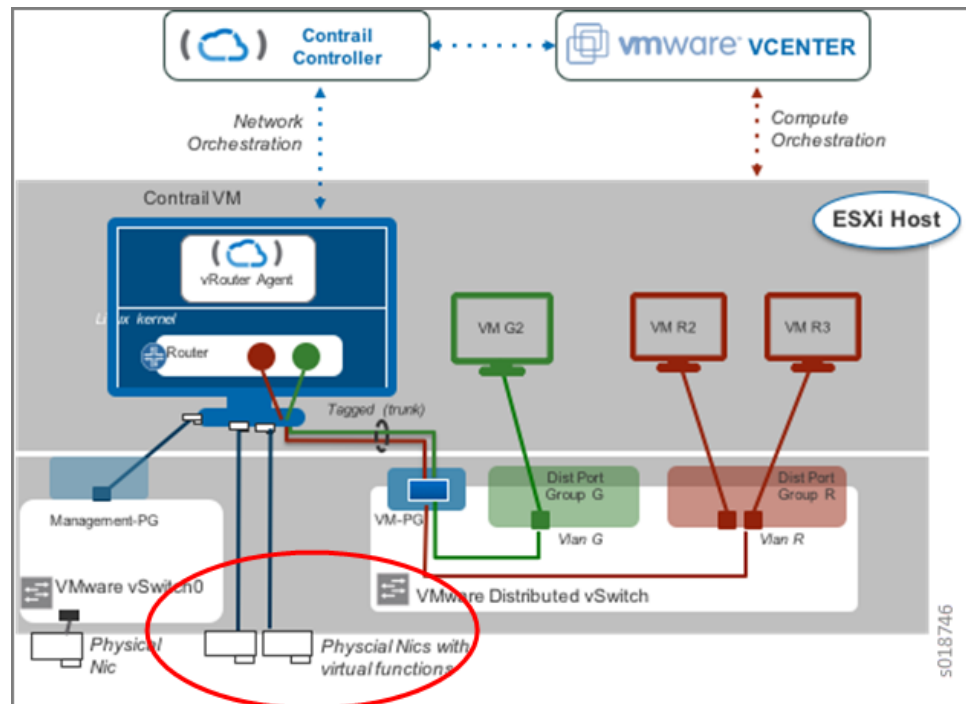
```

    "vmdk": "/root/vmdk_new/vmdk.tar"
  }

```

Figure 28 on page 113 shows an SR-IOV configuration with a bond **control_data** interface, which has multiple SR-IOV NICs.

Figure 28: SR-IOV With Bond Control Data Interface



For Bond interface-configuration specify multiple NICs in `sr_iov_nics`, and add required configuration for multi-interface and bond configuration in server JSON configuration as specified in Server-Manager documentation.

The cluster configuration:

```

"vcenter_servers": [
  {
    "server1": {
      "datacenters": {
        "i27_datacenter11": {
          "dv_switch_sr_iov": {
            "dv_port_group_sr_iov": {
              "dv_portgroup_name": "",
              "number_of_ports": "",
              "uplink": ""
            },
            "dv_switch_name": ""
          }
        }
      }
    }
  }
]

```

The server configuration:

```
"contrail_vm": {  
    "sr_iov_nics": {  
        "nics": ["vmnic0", "vmnic1"]  
    }  
    "vmdk": "/root/vmdk_new/vmdk.tar"  
}
```

**Related
Documentation**

- [Installing and Provisioning VMware vCenter with Contrail on page 97](#)
- [Sample JSON Configuration Files for vCenter with Containerized Contrail 4.0.1 and Greater](#)
- [Using the Contrail and VMWare vCenter User Interfaces to Manage the Network on page 119](#)

Sample Testbed.py Files for Contrail vCenter

- [Sample Testbed.py File for vCenter-Only Mode on page 114](#)
- [Sample Testbed.py File for vCenter-as-Compute Mode on page 116](#)

Sample Testbed.py File for vCenter-Only Mode

```
from fabric.api import env  
  
#Management ip addresses of hosts in the cluster  
host1 = 'user@10.xx.xx.xx' #Contrail Controller  
host2 = 'user@10.xx.xx.xx' #ContrailVM on ESXi  
  
#External routers if any  
ext_routers = []  
  
#Autonomous system number  
router_asn = 64512  
  
#Host from which the fab commands are triggered to install and provision  
host_build = 'user@10.xx.xx.xx'  
  
minimum_diskGB=32  
  
env.ntp_server = 'ntp.juniper.net'  
  
#Role definition of the hosts.  
env.roledefs = {  
    'all': [host1, host2],  
    'cfgm': [host1],  
    'control': [host1],  
    'compute': [host2],  
    'collector': [host1],  
    'webui': [host1],  
    'database': [host1],  
    'build': [host_build],  
}  
  
#Openstack admin password  
env.openstack_admin_password = 'secret123'  
env.orchestrator = 'vcenter' #other values are 'vcenter' default:openstack
```

```

#Disable multi-tenancy feature
multi_tenancy = False

env.password = 'secret'
#Passwords of each host
env.passwords = {
    host1: '<password>',
    host2: '<password>',
    host_build: '<password>',
}

#For reimage purpose
env.ostypes = {
    host1: 'ubuntu',
    host2: 'ubuntu',
}

#####
#vcenter provisioning
#server is the vcenter server ip
#port is the port on which vcenter is listening for connection
#username is the vcenter username credentials
#password is the vcenter password credentials
#auth is the authentication type used to talk to vcenter, http or https
#datacenter is the datacenter name we are operating on
#datacenter_mtu is the mtu size across the datacenter
#    optional, defaults to 1500
#cluster is the clustername we are operating on
#dvswitch section contains distributed switch related para,s
#    dv_switch_name is the name of the dvswitch
#dvportgroup section contains the distributed port group info
#    dv_portgroupname and the number of ports the group has
#####
env.vcenter_servers = {
    'vcenter1': {
        'server': '10.xx.xx.xx',
        'port': '443',
        'username': 'administrator@vsphere.local',
        'password': '<password>!',
        'auth': 'https',
        'datacenter': 'test_dc',
        'datacenter_mtu': '1500',
        'cluster': ['test_cluster'],
        'dv_switch': { 'dv_switch_name': 'test_dvswitch', },
        'dv_port_group': { 'dv_portgroup_name': 'test_dvportgroup',
                           'number_of_ports': '3',
                           },
    },
}

#####
# The compute vm provisioning on ESXI host
# This section is used to copy a vmdk on to the ESXI box and bring it up
# the contrailVM which comes up will be setup as a compute node with only
# vrouter running on it. Each host has an associated esxi to it.
#
# esxi_host information:
#   ip: the esxi ip on which the contrailvm(host/compute) runs
#   username: username used to login to esxi
#   password: password for esxi

```

```

# fabric_vswitch: the name of the underlay vswitch that runs on esxi
#                   optional, defaults to 'vswitch0'
# fabric_port_group: the name of the underlay port group for esxi
#                   optional, defaults to contrail-fab-pg'
# uplink_nic: the nic used for underlay
#             optional, defaults to None
# data_store: the datastore on esxi where the vmdk is copied to
# vcenter_server: the vcenter_server name which manages this esxi
# cluster: name of the cluster to which this esxi is added
# contrail_vm information:
#   mac: the virtual mac address for the contrail vm
#   host: the contrail_vm ip in the form of 'user@contrailvm_ip'
#   mode: the mode is 'openstack' or 'vcenter'
#         optional, defaults to env.orchestrator value
#   pci_devices: pci_devices information
#   nic: pci_id of the pass-through interfaces
#   sr_iov_nics: virtual functions enabled physical interface's name
#   vmdk: the absolute path of the contrail-vmdk used to spawn vm
#         optional, if vmdk_download_path is specified
#   vmdk_download_path: download path of the contrail-vmdk.vmdk used to spawn
vm
#                   optional, if vmdk is specified
#####
esxi_hosts = {
    'b4s4': {
        'ip': '10.xx.xx.xx',
        'username': 'root',
        'password': '<password>',
        'datastore': '/vmfs/volumes/',
        'vcenter_server': 'vcenter1',
        'cluster': 'test_cluster',
        'contrail_vm': {
            'mac': "00:50:56:05:ba:ba",
            'host': host2,
            'mode': "vcenter",
            'vmdk': "/tmp/ContrailVM-disk1.vmdk",
        }
    },
}

```

Sample Testbed.py File for vCenter-as-Compute Mode

```

from fabric.api import env

#Management ip addresses of hosts in the cluster
host1 = 'user@10.xx.xx.xx' #Contrail Controller
host2 = 'user@10.xx.xx.xx' #ContrailVM on ESXi
host3 = 'user@10.xx.xx.xx' #vcenter-compute
host4 = 'user@10.xx.xx.xx' #KVM Compute

#External routers if any
ext_routers = []

#Autonomous system number
router_asn = 64512

#Host from which the fab commands are triggered to install and provision
host_build = user@10.xx.xx.xx4'

```

```

minimum_diskGB=32

env.ntp_server = 'ntp.juniper.net'

#Role definition of the hosts.
env.roledefs = {
    'all': [host1, host2, host3, host4 ],
    'cfgm': [host1],
    'control': [host1],
    'compute': [host2, host4],
    'collector': [host1],
    'webui': [host1],
    'database': [host1],
    'openstack': [host1],
    'vcenter_compute': [host3 ],
    'build': [host_build],
}

#Openstack admin password
env.openstack_admin_password = 'secret123'

env.password = 'secret'
#Passwords of each host
env.passwords = {
    host1: '<password>',
    host2: '<password>',
    host3: '<password>',
    host4: '<password>',
    host_build: '<password>',
}

#For reimage purpose
env.ostypes = {
    host1: 'ubuntu',
    host2: 'ubuntu',
    host3: 'ubuntu',
    host4: 'ubuntu',
}

#To enable multi-tenancy feature
multi_tenancy = True

#####
#vcenter provisioning
#server is the vcenter server ip
#port is the port on which vcenter is listening for connection
#username is the vcenter username credentials
#password is the vcenter password credentials
#auth is the authentication type used to talk to vcenter, http or https
#datacenter is the datacenter name we are operating on
#datacenter_mtu is the mtu size across the datacenter
# optional, defaults to 1500
#cluster is the clustername we are operating on
#vcenter_compute is the nova-compute node for this vcenter server
#dvswitch section contains distributed switch related para,s
# dv_switch_name is the name of the dvswitch
#dvportgroup section contains the distributed port group info
# dv_portgroupname and the number of ports the group has
#####
env.vcenter_servers = {

```

```

        'vcenter1': {
            'server': '10.xx.xx.xx',
            'port': '443',
            'username': 'administrator@vsphere.local',
            'password': '<password>',
            'auth': 'https',
            'datacenter': 'test_dc',
            'datacenter_mtu': '1500',
            'cluster': ['test_cluster'],
            'vcenter_compute': '10.xx.xx.xx',
            'dv_switch': { 'dv_switch_name': 'test_dvswitch', },
            'dv_port_group': { 'dv_portgroup_name': 'test_dvportgroup',
                               'number_of_ports': '3',
                             },
        },
    },
}

#####
# The compute vm provisioning on ESXI host
# This section is used to copy a vm disk on to the ESXI box and bring it up
# the contrailVM which comes up will be setup as a compute node with only
# vrouter running on it. Each host has an associated esxi to it.
#
# esxi_host information:
#   ip: the esxi ip on which the contrailvm(host/compute) runs
#   username: username used to login to esxi
#   password: password for esxi
#   fabric_vswitch: the name of the underlay vswitch that runs on esxi
#                   optional, defaults to 'vswitch0'
#   fabric_port_group: the name of the underlay port group for esxi
#                      optional, defaults to contrail-fab-pg'
#   uplink_nic: the nic used for underlay
#               optional, defaults to None
#   data_store: the datastore on esxi where the vm disk is copied to
#   vcenter_server: the vcenter_server name which manages this esxi
#   cluster: name of the cluster to which this esxi is added
#   contrail_vm information:
#       mac: the virtual mac address for the contrail vm
#       host: the contrail_vm ip in the form of 'user@contrailvm_ip'
#       mode: the mode is 'openstack' or 'vcenter'
#             optional, defaults to env.orchestrator value
#       pci_devices: pci_devices information
#                   nic: pci_id of the pass-through interfaces
#                   sr_iov_nics: virtual functions enabled physical interface's name
#                   vm disk: the absolute path of the contrail-vm disk used to spawn vm
#                           optional, if vm disk_download_path is specified
#                   vm disk_download_path: download path of the contrail-vm disk used to spawn
#                                           vm
#                                           optional, if vm disk is specified
#####
esxi_hosts = {
    'b4s4': {
        'ip': '10.xx.xx.xx',
        'username': 'user',
        'password': '<password>',
        'datastore': '/vmfs/volumes/',
        'vcenter_server': 'vcenter1',
        'cluster': 'test_cluster',
        'contrail_vm': {
            'mac': '00:50:56:05:ba:ba',
            'host': host2,

```

```

        'mode': "vcenter",
        'vmdk': "/tmp/ContrailVM-disk1.vmdk",
    },
}

```

- Related Documentation**
- [Installing and Provisioning VMware vCenter with Contrail on page 97](#)
 - [Underlay Network Configuration for ContrailVM on page 106](#)

Using the Contrail and VMWare vCenter User Interfaces to Manage the Network

You can install Contrail to work with the VMware vCenter Server in various vSphere environments and use the Contrail user interface and the vCenter user interface to configure and manage the integrated Contrail system.

- [Overview: User Interfaces for Contrail Integration with VMware vCenter on page 119](#)
- [Feature Configuration for Contrail vCenter on page 120](#)
- [Creating a Virtual Machine on page 127](#)
- [Configuring the vCenter Network in Contrail UI on page 133](#)

Overview: User Interfaces for Contrail Integration with VMware vCenter

This topic shows how to use the Contrail user interface and the vCenter user interface to configure and manage features of a Contrail VMware integrated system.

The two user interfaces are available after installing the integrated Contrail system, see [“Installing and Provisioning VMware vCenter with Contrail” on page 97](#).

When Contrail is integrated with VMware vCenter, the following two user interfaces are used to manage and configure features of the system.

- [Contrail Administration User Interface on page 119](#)
- [Contrail vCenter User Interface on page 119](#)

Contrail Administration User Interface

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

To log in to the Contrail UI, use your Contrail server main IP address URL as follows:

https://<Contrail IP>:8080

Then log in using your registered Contrail account administrator credentials.

Contrail vCenter User Interface

The Contrail vCenter user interface (vCenter UI) is a subset of the Contrail administration UI. The Contrail vCenter UI provides a view of all of the virtual components within a Contrail vCenter project.

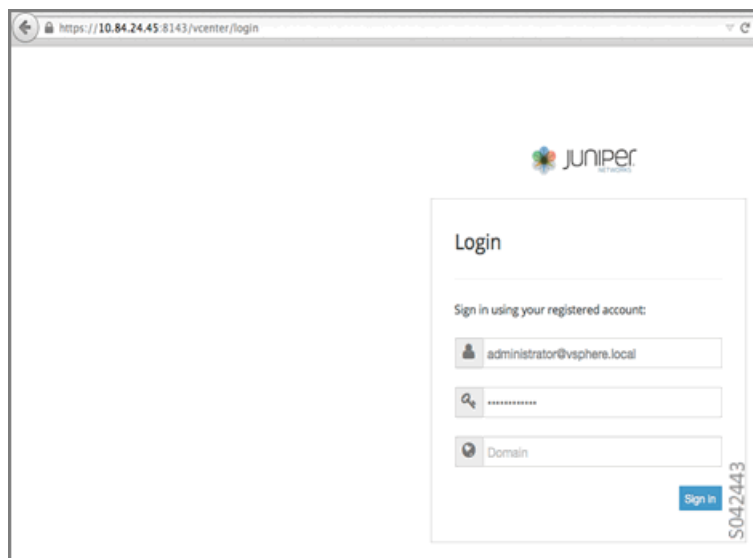


NOTE: This is applicable only to the vCenter-only mode.

To access the login page for the Contrail vCenter UI, use your Contrail IP address URL as follows:

`https://<Contrail URL>:8080/vcenter`

Then use the vCenter registered account log in name and password to access the Contrail vCenter UI, as in the following.



Upon successful login, the Contrail vCenter user interface is displayed, as in the following example.



Feature Configuration for Contrail vCenter

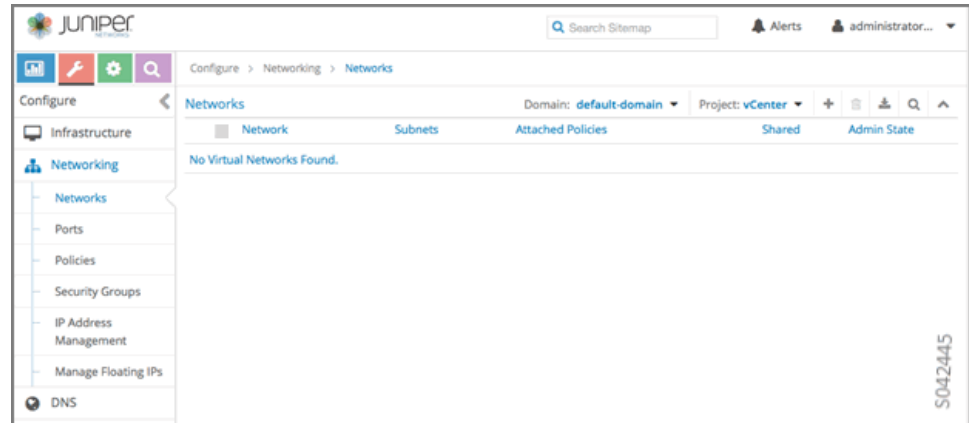
This section shows how to use the Contrail UI and the Contrail vCenter UI to configure features for the Contrail vCenter integrated system.

Creating a Virtual Network

This section describes how to create a virtual network using the Contrail UI and the Contrail vCenter UI.

Create Virtual Network – Contrail UI

After logging in to the Contrail UI, select **Configure > Networking > Networks** to access the Networks window.



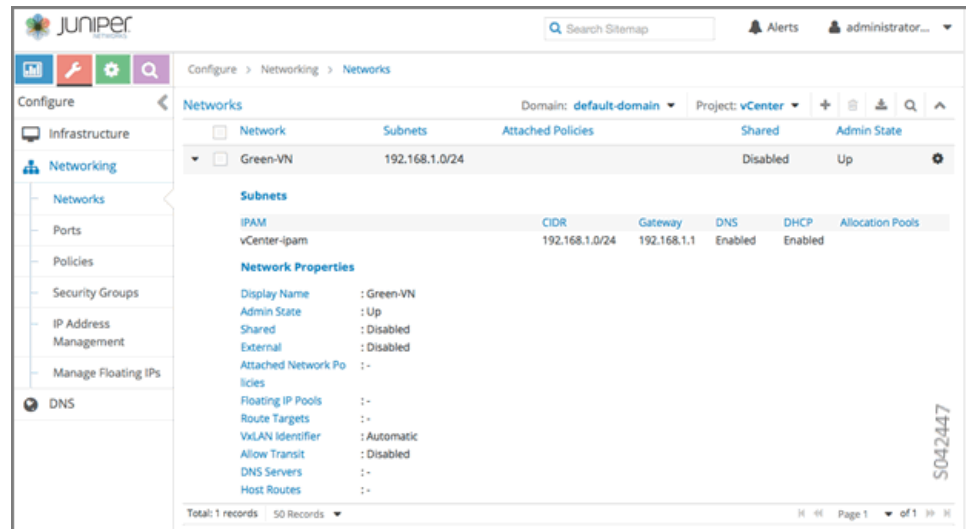
At Networks, click the plus icon (+) to access the **Create Network** window.

Complete the fields in the **Create Network** window. Provide a **Primary VLAN** value and a **Secondary VLAN** value as part of a **Private VLAN** configuration. **Private VLAN** pairs are configured on a Distributed Virtual Switch. Select the values for the Primary and Secondary VLANs from one of the configured, isolated, private-vlan pairs.

The following figure shows the creation of a virtual network named Green-VN.

Click **Save** to create the virtual network.

The virtual network just created (Green-VN) is displayed with its details, as in the following figure.



Create Virtual Networks – Contrail vCenter UI

You can also create a virtual network in the vCenter UI, and view and manage it from either the vCenter UI or the Contrail UI.



NOTE: This is applicable only to the vCenter-only mode.

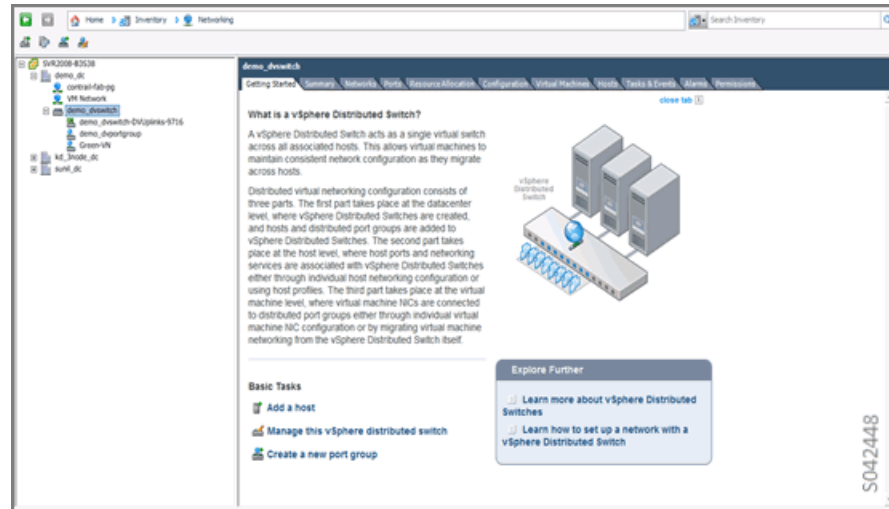
In vCenter, a virtual network is called a port group, which is part of a distributed switch.

Log in to the vCenter client UI (<https://<Contrail URL>:9443/vsphere-client>).

To start creating a virtual network (distributed port group), click the distributed virtual switch (**dvswitch**) on the left panel.

The following figure shows the *demo_dvswitch* has been selected for this example.

To create a virtual network (vCenter port group), at the bottom of the window, click **Create a new port group**.

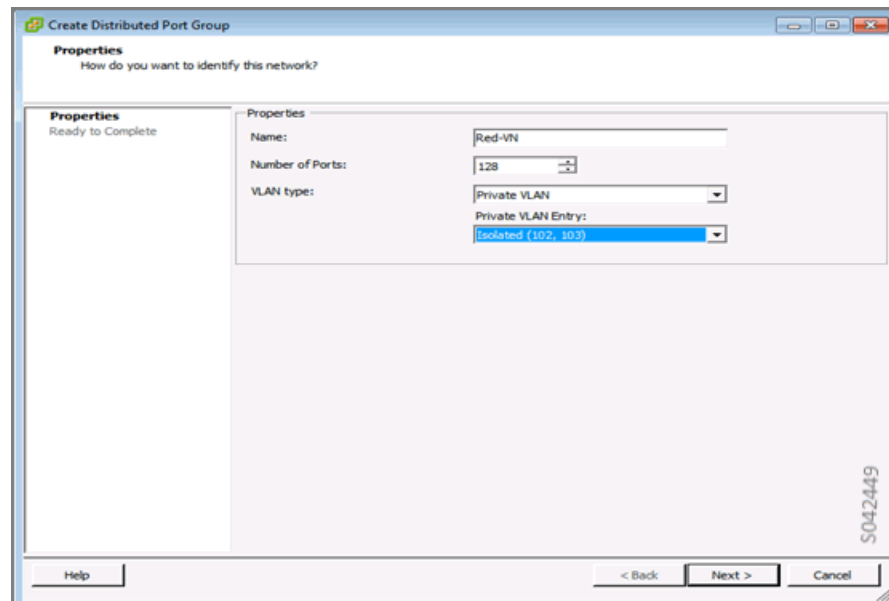


When you click **Create a new port group**, the **Create Distributed Port Group** window is displayed, as in the following figure.

Enter the name of the virtual network. Select the **VLAN type**, then select other details for the selected VLAN type.

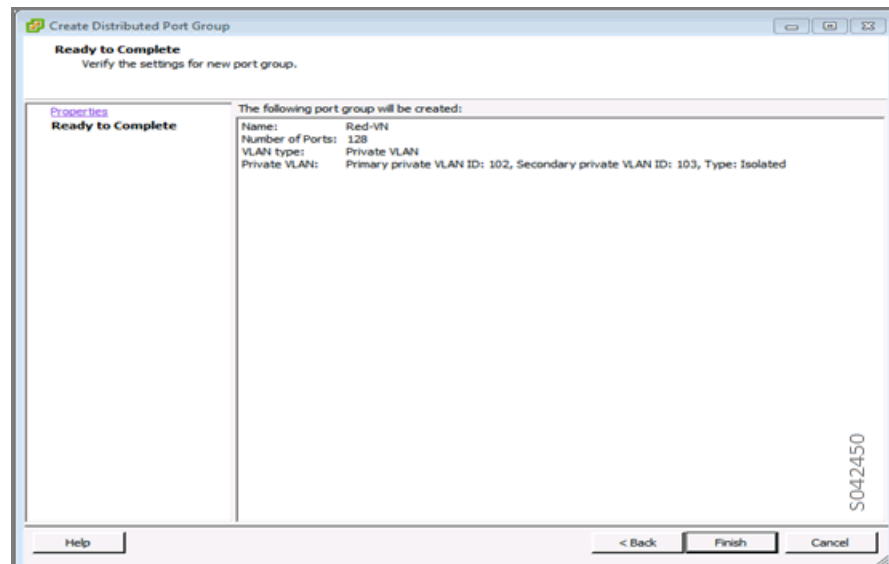
The following figure shows the **Create Distributed Port Group** window with the example creation of a virtual network named Red-VN, with a Private VLAN and isolated private VLAN ports 102, 103.

When you are finished, click **Next**.



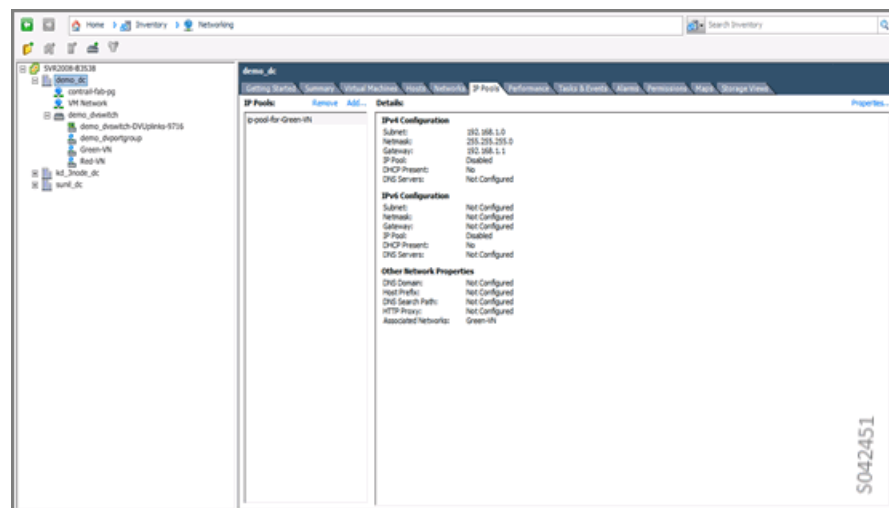
The **Ready to Complete** window is displayed, see the following figure. It shows the details entered for the virtual network (distributed port group).

If changes are needed, click **Back**. If the details are correct, click **Finish** to verify the port group details and complete its creation.



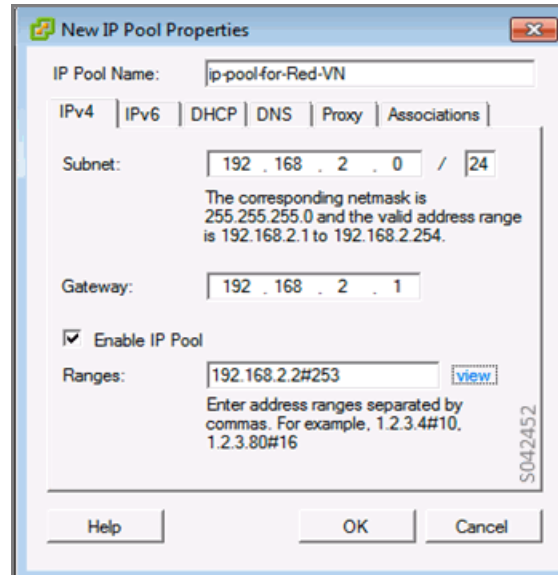
Next, create IP pools for the virtual network port group. Select the datacenter name in the left side panel, then click the **IP Pools** tab.

The following figure shows the **IP Pools** tab for the datacenter named demo_dc.



Near the top of the **IP Tools** window, click **Add** to open the **New IP Pool Properties** window, as in the following figure. The **IP Pool Properties** window has several tabs across the upper area. Ensure the **IPv4** tab is selected, and enter a name for the IP pool. Then enter the IP

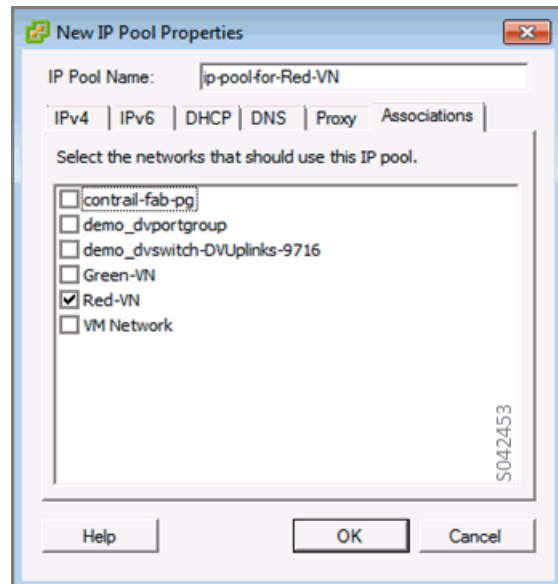
pool IPv4 details, including subnet, gateway, and IP address ranges. To enable IP address pools, select **Enable IP Pool**.



In the **New IP Pool Properties** window, click the **Associations** tab to select the networks that should use the IP address pool you are creating. This tab enables you to associate the IP pool with the port group.

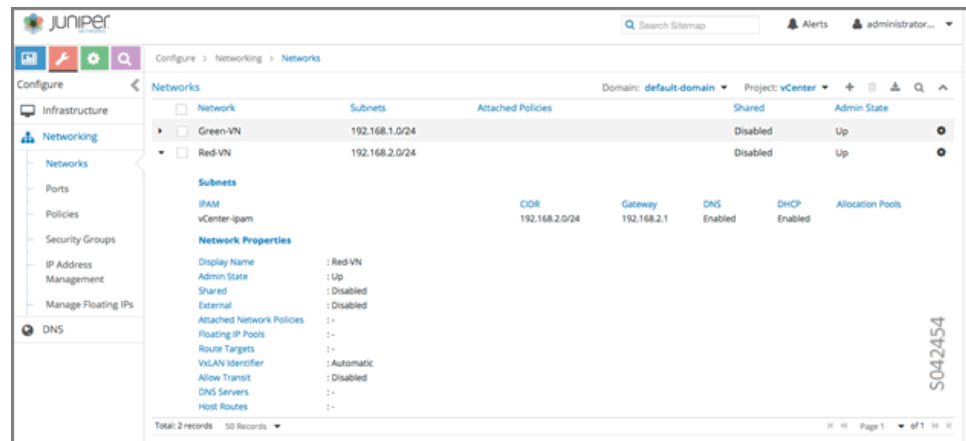
The following figure of the **Associations** tab shows that the IP pool being created should be associated with the virtual network port group named Red-VN.

When you are finished, click **OK**.



To verify that the virtual network is created and visible to Contrail, in the Contrail UI, select **Configure > Networking > Networks** to display Contrail network information.

The virtual network just created (Red-VN in this example) is displayed in the **Networks** window, see the following.

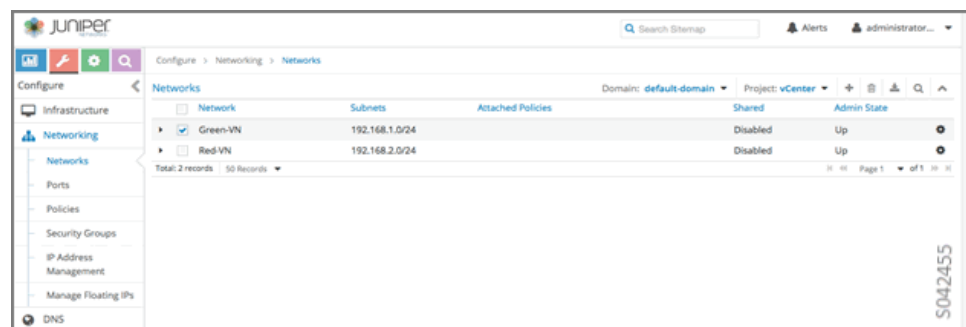


Delete Virtual Network – Contrail UI

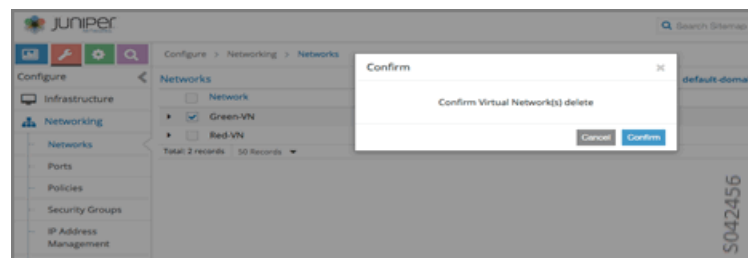
You can delete a virtual network in either the Contrail UI or in the vCenter UI. This section shows you how to delete a virtual network in the Contrail UI.

In the Contrail UI, select **Configure > Networking > Networks** to display Contrail network information.

Select the network you want to delete, then click the trashcan icon.



A Confirm window is displayed. Click **Confirm** to delete the selected network.



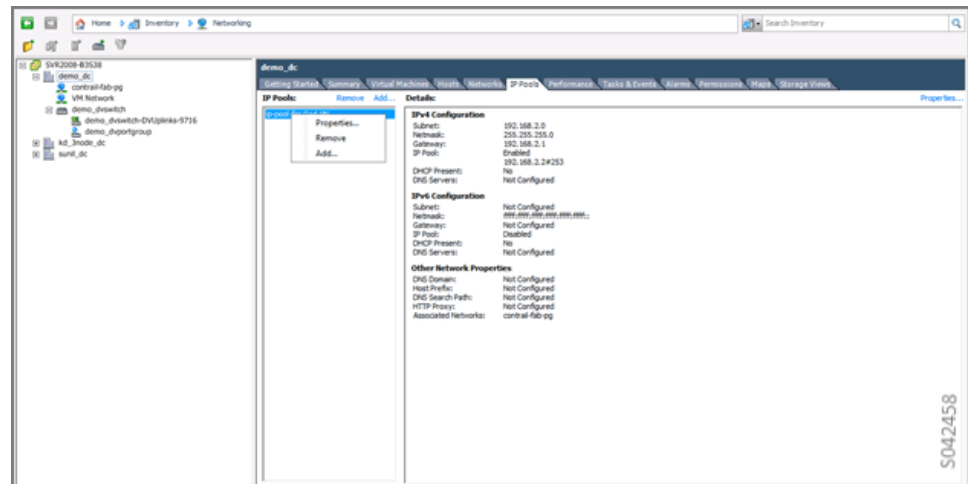
Delete Virtual Networks – vCenter UI

You can also delete a virtual network from the vCenter UI. From the vCenter UI, in the left side panel, right-click the port-group (virtual-network) you want to delete. In the menu, select **Delete** to delete the selected port group. An example is shown in the following.



When deleting a port group (virtual network) using the vCenter UI, you must also delete the IP pool associated with the port group. Select the **IP Pools** tab, and right click the name of the IP pool associated with the port group being deleted. From the menu, select **Remove** to delete the IP pool.

The following shows the deletion of the IP pool associated with the Red-VN from the vCenter UI.



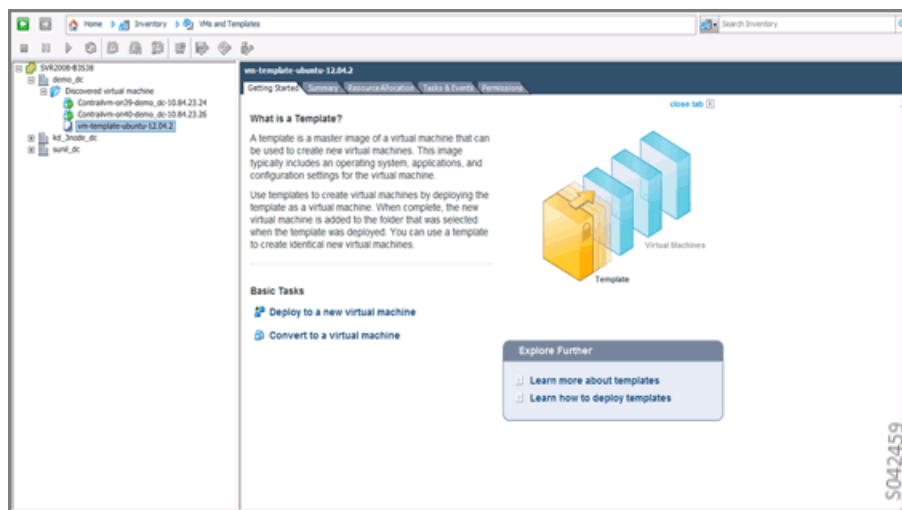
Creating a Virtual Machine

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

Create a Virtual Machine – vCenter UI

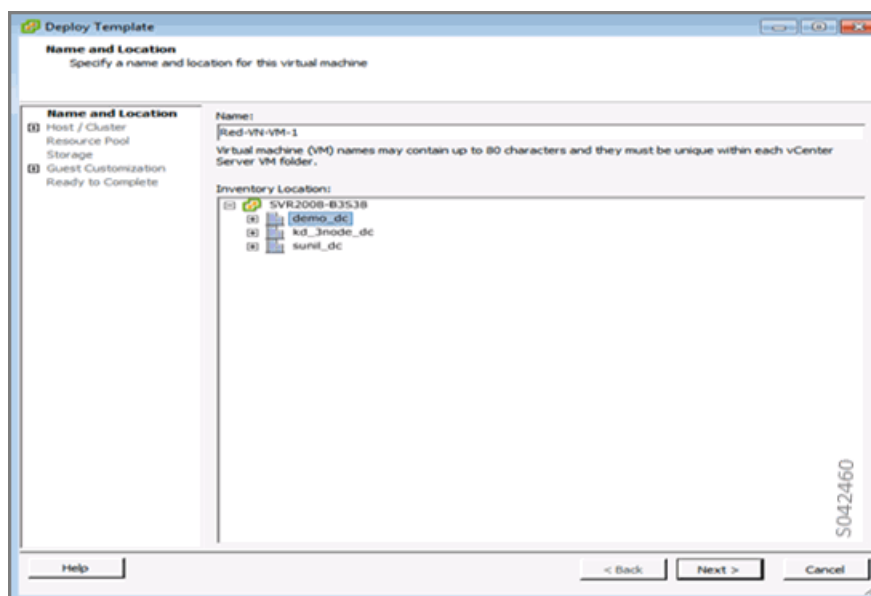
From the vCenter UI, select the virtual machine template from the left side panel. At the bottom of the right side pane, click **Deploy** to deploy a new virtual machine.

The following figure shows the **vm-template-ubuntu-12.04.2** virtual machine selected.



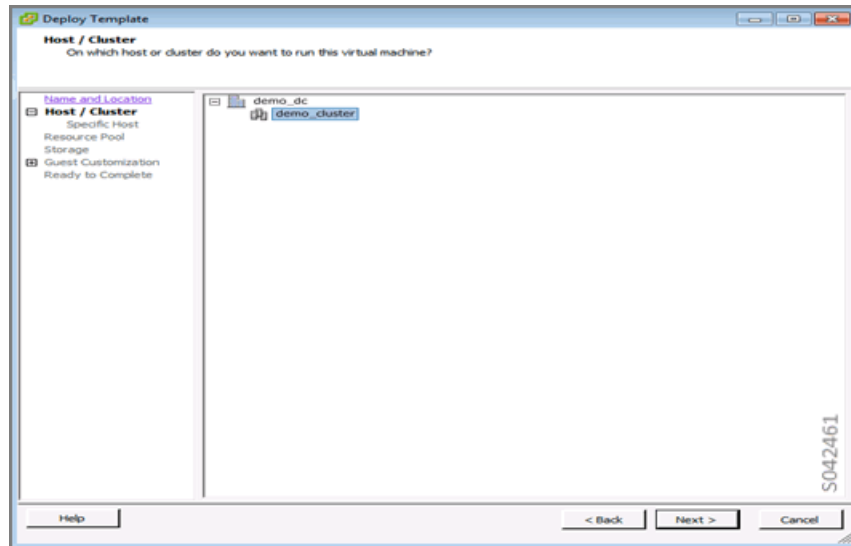
The **Deploy Template** Name and Location window is displayed, as in the following. Specify a name for the virtual machine and select the datacenter on which the virtual machine is to be spawned.

When you are finished, click **Next**.



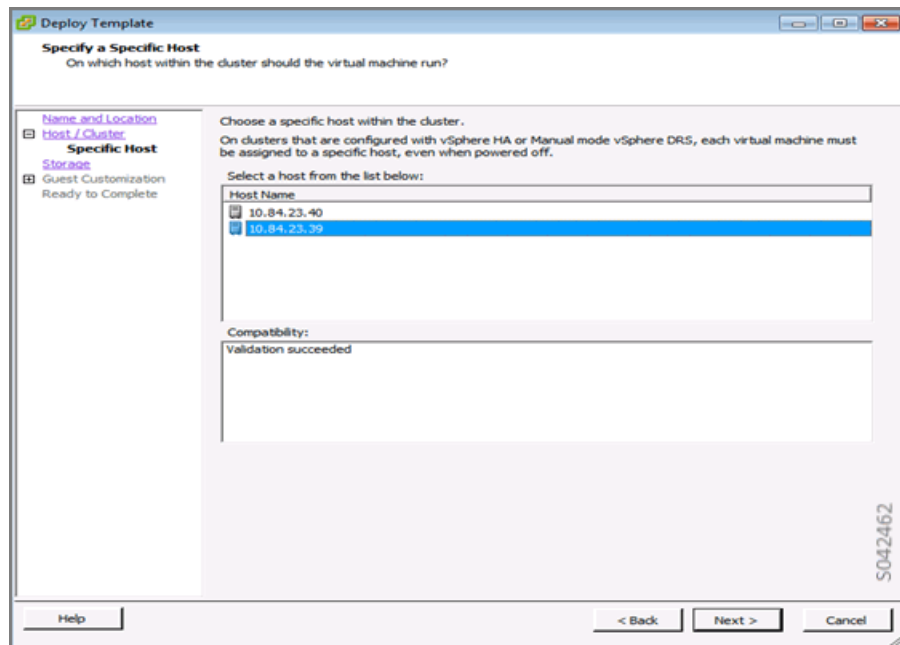
The **Host/Cluster** window is displayed, as in the following. Select the cluster on which to spawn the virtual machine.

When you are finished, click **Next**.



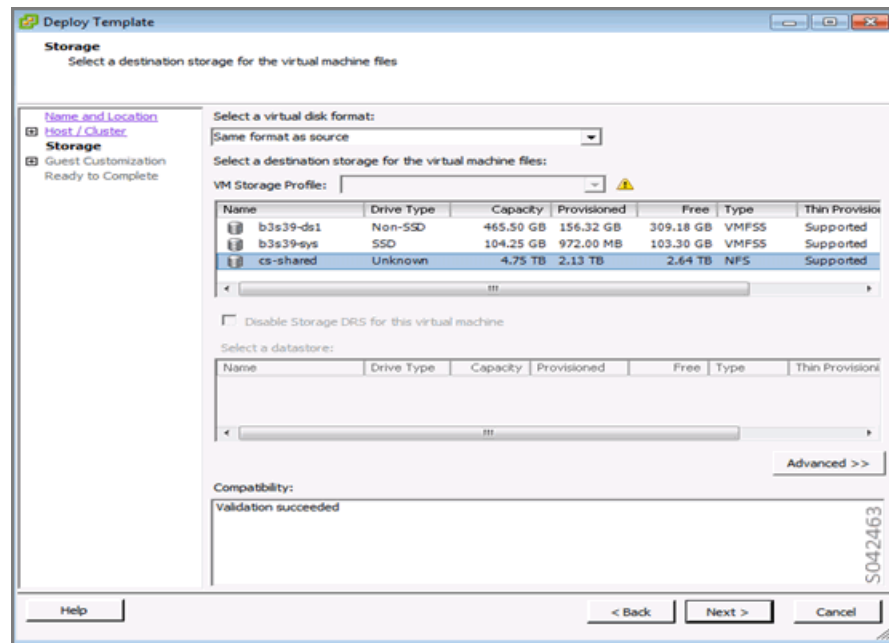
The **Specify a Specific Host** window is displayed, as in the following. Select the ESXi host on which to spawn the virtual machine.

When you are finished, click **Next**.



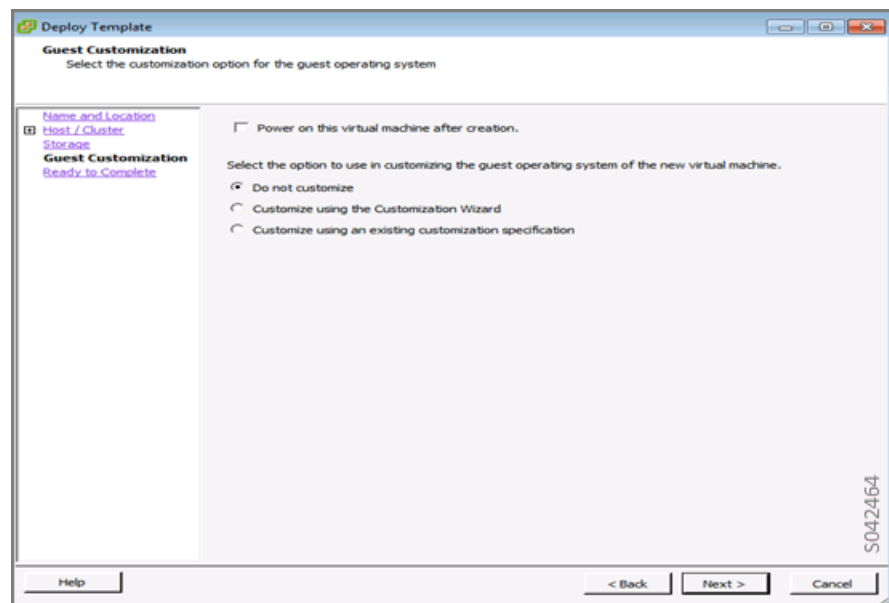
In the **Storage** window, select the destination storage location for the virtual machine.

When you are finished, click **Next**.



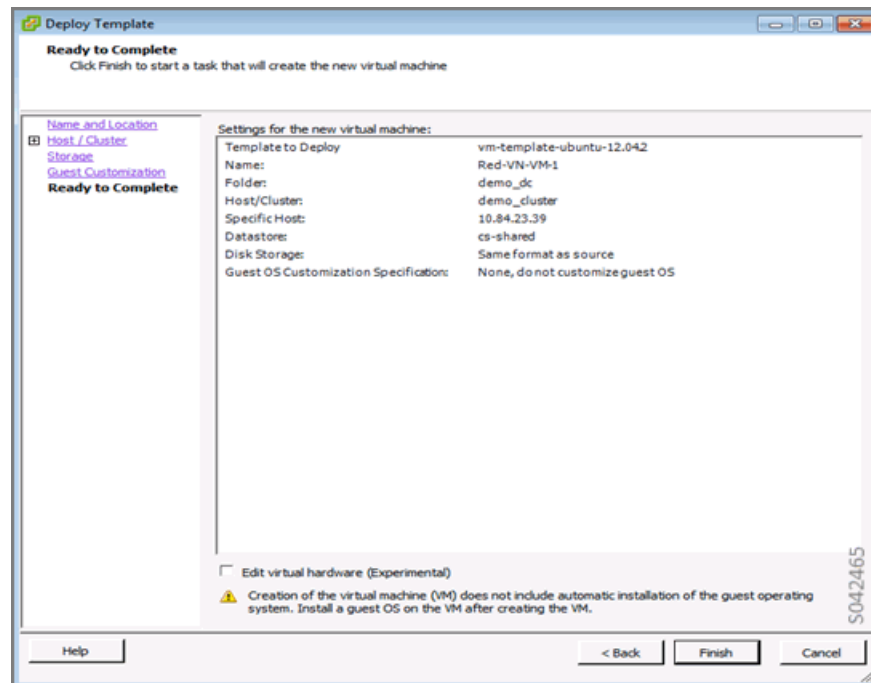
On the **Guest Customization** window, the typical selection is Do not customize. Select **Do not customize**.

When you are finished, click **Next**.



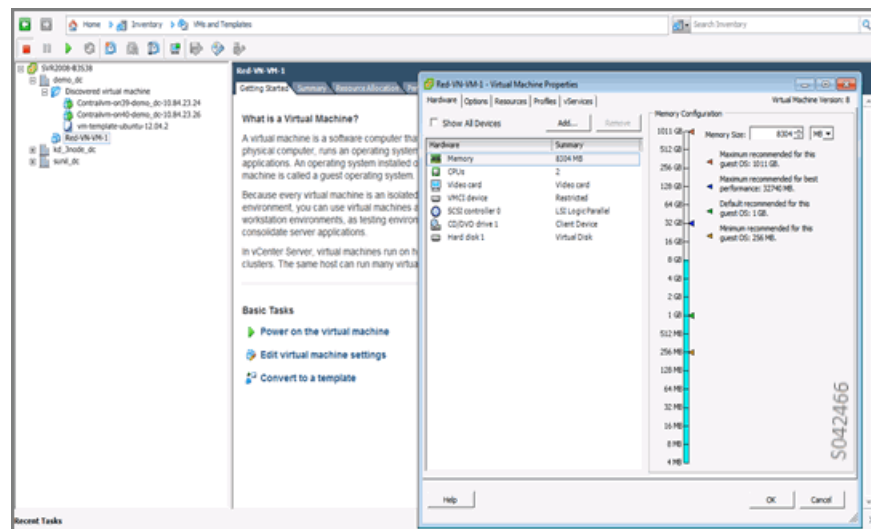
On the **Ready to Complete** window, review all of the virtual machine definitions that you have selected for the template.

If all the selections are correct, click **Finish**. This spawns the virtual machine.



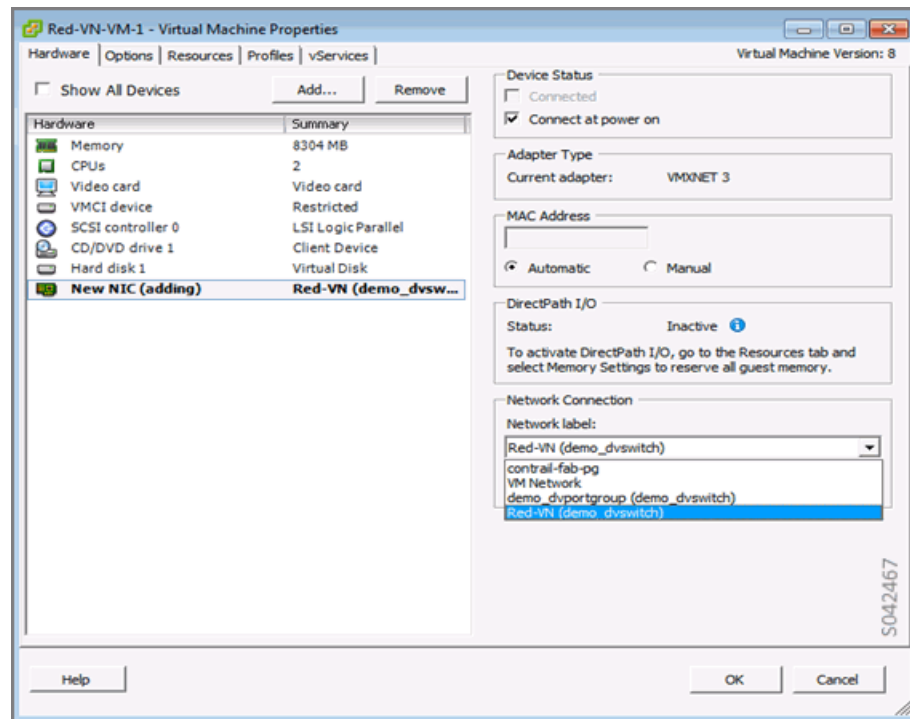
To complete the settings for the virtual machine, select the virtual machine to be edited in the left column of the main window of the vCenter UI. Then click **Edit virtual machine settings**.

The **Virtual Machine Properties** window is displayed, as in the following. From here you can update the virtual machine properties.

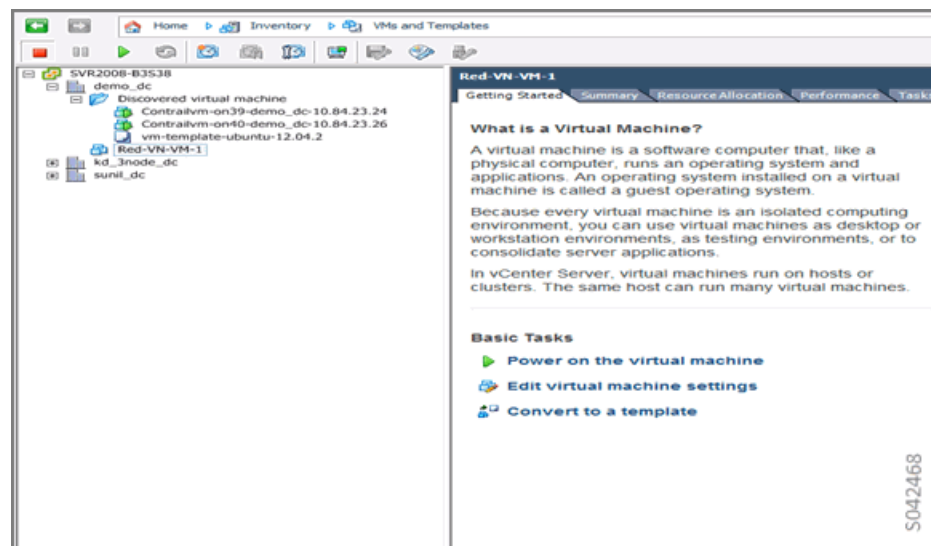


Click the **Hardware** tab in the **Virtual Machine Properties** window. Next, click **Add** to add a NIC and select the appropriate network. Select **Connect at power on**, as in the following.

When you are finished, click **OK**.



You are returned to the main vCenter UI window. Select the **Getting Started** tab. Select **Power on the virtual machine**. The virtual machine launches.



Once the virtual machine is launched, you can view it from the Contrail UI. Select **Monitor > Networking > Instances**. The virtual machines are displayed in the **Instances Summary** window, as in the following.

Instance	UUID	Virtual Network	Interface	vRouter	IP Address	Floating IPs (In/Out)	Traffic (In/Out) (Last 1 hr)
Red-VN-VM-1	5039e121-36b2-4f54-9c38-2b251f64b1	Red-VN (vCenter)	1	ContrailVM	IPv4: 192.168.2.254		2.72 KB / 2.04 KB

You can also see real-time running information for the virtual machine in the vCenter UI. Select the virtual machine and the **Console** tab. Real-time information is displayed, including ping statistics, as in the following.

```

UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
RX packets:111 errors:0 dropped:0 overruns:0 frame:0
TX packets:21 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:1750 (1.7 KB)  TX bytes:2008 (2.0 KB)

io
Link encap:Local Loopback
inet addr:127.0.0.1  Mask:255.0.0.0
inet6 addr: ::1/128 Scope:Host
UP LOOPBACK RUNNING  MTU:65536  Metric:1
RX packets:0 errors:0 dropped:0 overruns:0 frame:0
TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:0
RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

root@ubuntu-server-12:~# ping 192.168.2.1
PING 192.168.2.1 (192.168.2.1) 56(84) bytes of data:
64 bytes from 192.168.2.1: icmp_seq=1 ttl=64 time=0.523 ms
64 bytes from 192.168.2.1: icmp_seq=2 ttl=64 time=0.413 ms
64 bytes from 192.168.2.1: icmp_seq=3 ttl=64 time=0.296 ms
^C
--- 192.168.2.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1999ms
rtt min/avg/max/mdev = 0.296/0.410/0.523/0.095 ms
root@ubuntu-server-12:~#

```

Configuring the vCenter Network in Contrail UI

The following items can be configured for the vCenter network by using the Contrail UI.

- Network policy is configured by using the Contrail UI.
- Security policy is configured by using the Contrail UI.
- Public networks, floating IP address pools, and floating IP addresses are configured using the Contrail Administrator UI.

When you configure a virtual network using the administrator UI, the network is a Contrail-only network. No resources are consumed on vCenter to implement this type of network. You can configure a floating IP address pool on the network, allocate floating IP addresses, and associate floating IP addresses to virtual machine interfaces (ports).

Related Documentation

- [Installing and Provisioning VMware vCenter with Contrail on page 97](#)

CHAPTER 7

Using Contrail with Red Hat

- [Deploying Contrail with Red Hat OpenStack Platform Director 10 on page 135](#)

Deploying Contrail with Red Hat OpenStack Platform Director 10

This document explains how to integrate a Contrail 3.2 through Contrail 4.1 installation with RedHat OpenStack Platform Director 10.

- [Overview on page 135](#)
- [TripleO Features on page 135](#)
- [Deployment Tools on page 137](#)
- [Preparing the Environment for Deployment on page 137](#)
- [Deploying an OSPD-10 Overcloud on page 141](#)
- [Configuring the Overcloud on page 144](#)
- [Sample Heat Templates for NICs on page 154](#)
- [What are NIC Templates? on page 162](#)
- [How NIC Templates Work on page 162](#)
- [Contrail NIC Templates on page 164](#)
- [NIC Templates for Compute Nodes on page 168](#)

Overview

RedHat OpenStack Platform provides an installer named Director (RHOSPD). The Red Hat Director installer is based on the OpenStack project TripleO (OOO, OpenStack on OpenStack). TripleO is an open source project that uses features of OpenStack to deploy a fully functional, tenant-facing OpenStack environment.

You can use TripleO and Director to deploy a Red Hat cloud environment integrated with Contrail.

TripleO Features

TripleO uses the concepts of undercloud and overcloud. TripleO sets up an undercloud, an operator-facing deployment cloud that contains the OpenStack components needed to deploy and manage an overcloud, a tenant-facing cloud that hosts user workloads.

The overcloud is the deployed solution that can represent a cloud for any purpose, such as production, staging, test, and so on. The operator can select to deploy to their environment any of the available overcloud roles, such as controller, compute, and the like.

TripleO leverages existing core components of OpenStack including Nova, Ironi, Neutron, Heat, Glance, and Ceilometer to deploy OpenStack on bare metal hardware.

- Nova and Ironi are used in the undercloud to manage the bare metal instances that comprise the infrastructure for the overcloud.
- Neutron is used to provide a networking environment in which to deploy the overcloud.
- Glance stores machine images.
- Ceilometer collects metrics about the overcloud.

For more information about TripleO architecture, see

<https://docs.openstack.org/developer/tripleo-docs/introduction/architecture.html>

Composable Roles

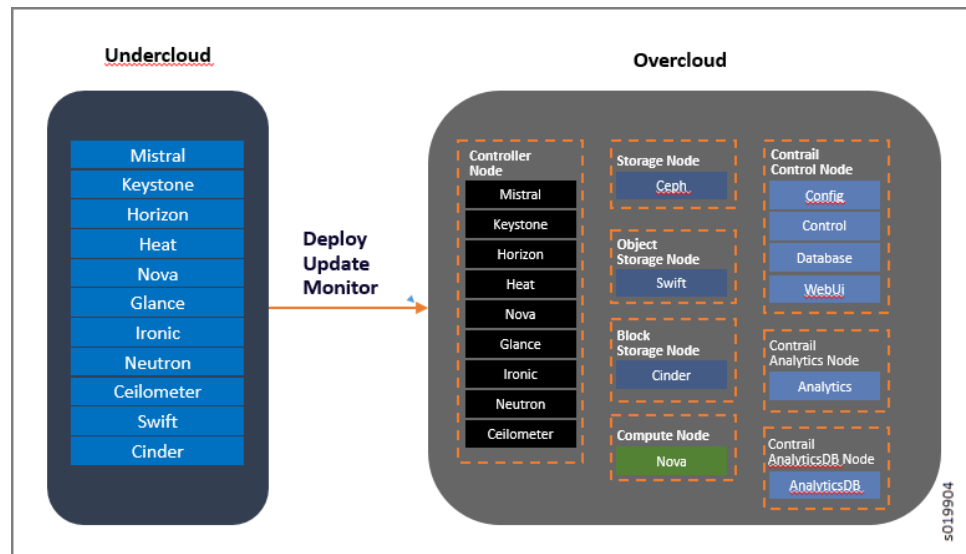
TripleO enables composable roles. Each role is a group of services that are defined in Heat templates. Composable roles gives the operator the flexibility to add and modify roles as needed.

The following are the Contrail roles used for integrating Contrail to the overcloud environment:

- Contrail Controller
- Contrail Analytics
- Contrail Analytics Database
- Contrail-TSN
- Contrail-DPDK

[Figure 29 on page 137](#) shows the relationship and components of an undercloud and overcloud architecture for Contrail.

Figure 29: Undercloud and Overcloud with Roles



Deployment Tools

Deployment to physical servers or virtual machines occurs by means of collaboration between Heat, Nova, Neutron, Glance, and ironic.

One nested Heat stack is deployed from the undercloud. The Heat stack has various Nova instances. The Nova instances are the overcloud roles. The definitions for all roles are provided in the Heat templates.

To deploy the stack, Heat makes successive calls to Nova, OpenStack's compute service controller. Nova depends on ironic, which, by this stage in the process, has acquired an inventory of introspected hardware.

If configured, Nova flavors can act as a constraint, influencing the range of machines that can be selected for deployment by the Nova scheduler. For each request to deploy a new node with a specific role, Nova filters the list of available nodes, ensuring that the selected nodes meet the hardware requirements.

When the target node has been selected, ironic performs the provisioning of the node: ironic retrieves from Glance the OS image associated with the role, causes the node to boot a deployment ramdisk, and, in a typical case, exports the node's local disk over iSCSI so that the disk can be partitioned and have the OS image written onto it by the ironic conductor.

Preparing the Environment for Deployment

- [Preparing for the Contrail Roles on page 138](#)
- [Preparing for the Underlay Network on page 138](#)
- [Preparing for the Provisioning Network on page 139](#)
- [Network Isolation on page 139](#)

The overcloud roles can be deployed to bare metal servers or to virtual machines (VMs). The compute nodes must be deployed to bare metal systems.

Ensure your environment is prepared for the Red Hat deployment. Refer to Red Hat documentation:

https://access.redhat.com/documentation/en-us/red_hat_opensstack_platform/10/html-single/director_installation_and_usage

Preparing for the Contrail Roles

Ensure the following requirements are met for the Contrail nodes per role.

- Non-high availability: A minimum of four overcloud nodes are needed for control plane roles for a non-high availability deployment:
 - 1x contrail-config (includes Contrail control)
 - 1x contrail-analytics
 - 1x contrail-analytics-database
 - 1x OpenStack controller
- High availability: A minimum of 12 overcloud nodes are needed for control plane roles for a high availability deployment:
 - 3x contrail-config (includes Contrail control)
 - 3x contrail-analytics
 - 3x contrail-analytics-database
 - 3x OpenStack controller
 - If the control plane roles will be deployed to VMs, use 3 separate physical servers and deploy one role of each kind to each physical server.

RHOSP Director expects the nodes to be provided by the administrator, for example, if you are deploying to VMs, the administrator must create the VMs before starting with deployment.

Preparing for the Underlay Network

Refer to Red Hat documentation for planning and implementing underlay networking, including the kinds of networks used and the purpose of each:

- https://access.redhat.com/documentation/en-us/red_hat_opensstack_platform/10/html-single/director_installation_and_usage/#sect-Planning_Networks
- https://access.redhat.com/documentation/en-us/red_hat_opensstack_platform/10/html-single/director_installation_and_usage/#sect-Networking_Requirements

At a high level, every overcloud node must support IPMI.

Refer to “Requirements for Deploying to VMs” in this document if you are deploying to VMs.

Preparing for the Provisioning Network

Ensure the following requirements are met for the provisioning network.

- One NIC from every machine must be in the same broadcast domain of the provisioning network, and it should be the same NIC on each of the overcloud machines. For example, if you use the second NIC on the first overcloud machine, you should use the second NIC on each additional overcloud machine.

During installation, these NICs will be referenced by a single name across all overcloud machines.

- The provisioning network NIC should not be the same NIC that you are using for remote connectivity to the undercloud machine. During the undercloud installation, an Open vSwitch bridge will be created for Neutron and the provisioning NIC will be bridged to the Open vSwitch bridge. Consequently, connectivity would be lost if the provisioning NIC was also used for remote connectivity to the undercloud machine.
- The provisioning NIC on the overcloud nodes must be untagged.
- You must have the MAC address of the NIC that will PXE boot the IPMI information for the machine on the provisioning network. The IPMI information will include such things as the IP address of the IPMI NIC and the IPMI username and password.
- All of the networks must be available to all of the Contrail roles and computes.

Network Isolation

TripleO enables configuration of isolated overcloud networks. Using this approach, it is possible to host traffic in isolated networks for specific types of network traffic, such as tenants, storage, API, and the like. This enables assigning network traffic to specific network interfaces or bonds.

When isolated networks are configured, the OpenStack services are configured to use the isolated networks. If no isolated networks are configured, all services run on the provisioning network.

The following networks are typically used when using network isolation topology:

- Provisioning---for the undercloud control plane
- Internal API--- for OpenStack internal APIs
- Tenant
- Storage
- Storage Management
- External
 - Floating IP---Can either be merged with external or can be a separate network.
- Management

Templates for Network Isolation

Use the following template files to enable network isolation:

- **environments/network-isolation.yaml**

Contains the path of templates that need to be included to create various Neutron networks and ports

- **environments/contrail/contrail-net.yaml**

Contains the subnet/mask, allocation pool, default gateway IP information. Make changes to this file to configure the subnets for your setup.

- **environments/contrail/contrail-nic-config.yaml**

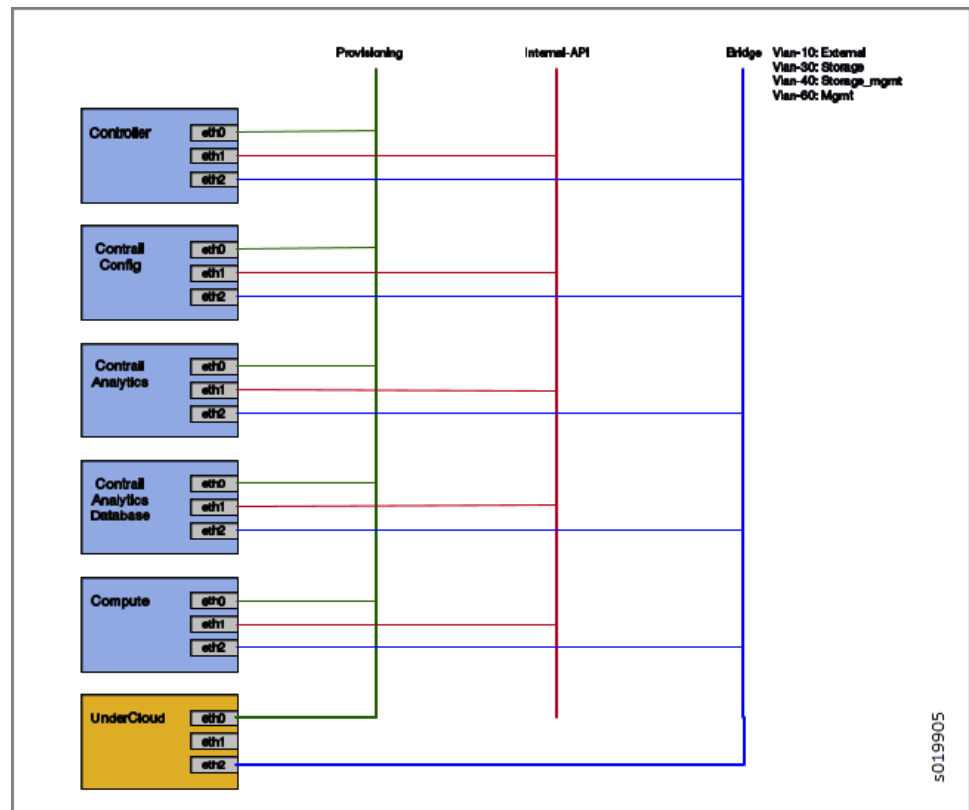
Defines the NICs that the overcloud VMs will use for each of the networks. Change the contents of this template as needed for your environment.

Features of the default configuration include:

- The first NIC is used for the control plane provisioning network.
- The second NIC is used for the internal API network.
- The third NIC uses multiple VLANs to provide for the rest of the networks:
 - VLAN-10: External network
 - VLAN-30: Storage network
 - VLAN-40: Storage management network
 - VLAN-50: Tenant network
 - VLAN-60: Management network
 - VLAN-XXX: Floating network (if separate from external network)

[Figure 30 on page 141](#) shows the network connectivity for the overcloud roles when you use the default Heat templates. Fig . In [Figure 30 on page 141](#), the vertical lines depict the underlay, which could be a switch. The underlay connectivity must be prepared before starting the deployment. The undercloud must have reachability in the provisioning network and the external networks.

Figure 30: Network Isolation Model



Deploying an OSPD-10 Overcloud

When the requirements for the environment are met, you are ready to start deploying.

- Installing the Undercloud on page 141
- Configuring Undercloud and Overcloud on page 142
- Defining Nodes with Ironic on page 142

Installing the Undercloud

Use the Red Hat OS Director to install the undercloud after the environment has been prepared. You'll need Red Hat credentials, such as account, password, pool, and the like, to register the undercloud and overcloud nodes.

Follow procedures in the Red Hat documentation to install an undercloud:

https://access.redhat.com/documentation/en-us/red_hat_openshift_platform/10/html-single/director_installation_and_usage/#chap-Installing_the_Undercloud

Configuring Undercloud and Overcloud

After the undercloud is installed, you can use the following procedures to change parameters in the **undercloud.conf** file to match your local deployment.

1. Configure the undercloud.

```
cp /usr/share/instack-undercloud/undercloud.conf.sample ~/undercloud.conf
vi ~/undercloud.conf
```

2. Install the undercloud OpenStack.

```
openstack undercloud install
```

3. Source the undercloud credentials.

```
source ~/stackrc
```

4. Get overcloud images.

```
sudo yum install rhosp-director-images rhosp-director-images-ipa
mkdir ~/images
cd ~/images
```

5. Upload overcloud images.

```
for i in /usr/share/rhosp-director-images/overcloud-full-latest-10.0.tar
/usr/share/rhosp-director-images/ironic-python-agent-latest-10.0.tar; do tar -xvf $i;
done
openstack overcloud image upload --image-path /home/stack/images/
cd ~
```

Defining Nodes with Ironic

The properties of the overcloud nodes and VMs are in the **instackenv.json** file, which is imported to Ironic.

This procedure shows how to define nodes with Ironic.

1. Define nodes in **instackenv.json**.

```
vi ~/instackenv.json
```

- A password-less SSH must be enabled on all hosts on which overcloud VMs will be spawned.
- If you need definitive node placement, assign the appropriate capabilities in the node definition in **instackenv.json**.

For more information about using **instackenv.json**, see Red Hat documentation:

- https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/10/html-single/director_installation_and_usage/#sect-Registering_Nodes_for_the_Overcloud
- https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/10/html-single/director_installation_and_usage/#sect-Registering_Nodes_for_the_Overcloud

A sample **instackenv.json** is included later in this topic.

2. Import nodes to Ironic.

```
openstack baremetal import --json ~/instackenv.json
```

3. Configure boot mode. This assigns the kernel and ramdisk image to each Ironic node.

```
openstack baremetal configure boot
```

4. Activate node introspection.

Introspection boots each Ironic node over the PXE network and is used to collect hardware data for the nodes. The capabilities and profile of each node is determined at this step. Because this step includes pushing an image to each of the overcloud roles, successful completion of Ironic introspection also means that the underlay configuration is valid on the provisioning network.



NOTE: Make sure that the maximum transmission unit (MTU) is consistent across all of the networks to prevent any issues.

```
for node in $(openstack baremetal node list -c UUID -f value) ; do openstack baremetal node manage $node ; done
```

```
openstack overcloud node introspect --all-manageable --provide
```

5. Perform node profiling.

If you provided capabilities for overcloud nodes, create the corresponding flavors at this time. Each overcloud role can be assigned a certain Nova-flavor in the Heat templates. You can provide details such as memory, disk-size, number of CPUs, and so on. At the time of deploying a role, Director tries to find an Ironic node that has the capabilities listed in the flavor. This is the way in which you can control node placement.

```
openstack flavor create <flavor-name> --ram <RAM> --vcpus <CPUs> --disk <disk-size>
openstack flavor set --property "capabilities:boot_option"="local" --property
"capabilities:profile"="<capability-name>" <flavor-name>
```

Configuring the Overcloud

- [Get Contrail Components on page 144](#)
- [Configure NICs for Overcloud Networking on page 145](#)
- [Assign Addresses and Credentials on page 146](#)
- [Deploying the Overcloud on page 147](#)
- [Sample instackenv.json on page 148](#)
- [Adding a New Physical Compute Node on page 153](#)

Get Contrail Components

This procedure provides the components needed to integrate Contrail with Director, including adding a repo that hosts Contrail packages and providing Heat templates and corresponding Puppet modules.

1. Create a Contrail repo.

A Contrail repo is needed to make sure that the overcloud Contrail roles can install the Contrail packages. The Contrail repo can be hosted on the undercloud or on any machine that is accessible from the overcloud nodes on the provisioning network.

```
sudo mkdir /var/www/html/contrail sudo tar zxvf
~/contrail-install-packages_<package>.tgz -C /var/www/html/contrail/
```

2. Upload Puppet modules to Swift.

Install the RPMs for Puppet modules in the directory:
home/stack/usr/share/openstack-puppet/modules/.

This folder must contain the Puppet modules necessary to successfully install and start Contrail services in the overcloud roles.

Use the command **upload-swift-artifacts** to make sure that these modules get uploaded on the overcloud nodes during deployment. All of the commands are executed as user **stack**.

```
cd /var/www/html/contrail

yum localinstall contrail-tripleo-puppet-<version>.el7.noarch.rpm
puppet-contrail-<version>.el7.noarch.rpm

mkdir -p ~/usr/share/openstack-puppet/modules/contrail

cp -R /usr/share/openstack-puppet/modules/contrail/*
~/usr/share/openstack-puppet/modules/contrail/

mkdir -p ~/usr/share/openstack-puppet/modules/tripleo
```



```
cp -R /usr/share/contrail-tripleo-puppet/*
~/usr/share/openstack-puppet/modules/tripleo

cd ~

tar czvf puppet-modules.tgz ~/usr/

upload-swift-artifacts -f puppet-modules.tgz
```

3. Get TripleO Heat templates.

```
cp -r /usr/share/openstack-tripleo-heat-templates/ ~/tripleo-heat-templates

cd /var/www/html/contrail

yum localinstall contrail-tripleo-heat-templates-<version>.el7.noarch.rpm

cp -r /usr/share/contrail-tripleo-heat-templates/environments/contrail
~/tripleo-heat-templates/environments

cp -r /usr/share/contrail-tripleo-heat-templates/puppet/services/network/*
~/tripleo-heat-templates/puppet/services/network
```

4. Update the **contrail-services.yaml**.

The **contrail-services.yaml** is the main administrator-facing Heat template. Provide the correct URL for the Contrail repo that you created, the flavor for overcloud roles, the count for overcloud roles, and other various environment-specific parameters such as DNS-server, NTP server, and the like.

```
vi ~/tripleo-heat-templates/environments/contrail/contrail-services.yaml
```

You must set the value for **ContrailVersion** to 3.

Configure NICs for Overcloud Networking

Use this information to configure the NICs for your system.

Overcloud Networking—Multiple NICs

```
vi ~/tripleo-heat-templates/environments/contrail/contrail-net.yaml
vi ~/tripleo-heat-templates/environments/contrail/contrail-nic-config-compute.yaml
vi ~/tripleo-heat-templates/environments/contrail/contrail-nic-config.yaml
```

Overcloud Networking—Multiple NICs with Bond and VLAN

```
vi ~/tripleo-heat-templates/environments/contrail/contrail-net-bond-vlan.yaml
vi
~/tripleo-heat-templates/environments/contrail/contrail-nic-config-compute-bond-vlan.yaml
vi ~/tripleo-heat-templates/environments/contrail/contrail-nic-config-vlan.yaml
```

Overcloud Networking—Single NIC

```
vi ~/tripleo-heat-templates/environments/contrail/contrail-net-single.yaml
vi
```

```
~/tripleo-heat-templates/environments/contrail/contrail-nic-config-compute-single.yaml
vi ~/tripleo-heat-templates/environments/contrail/contrail-nic-config-single.yaml
```

Assign Addresses and Credentials

1. Assign static IP addresses.

Use the template `ips-from-pool-all.yaml` to provide static IP addresses for the overcloud nodes.

```
vi ~/tripleo-heat-templates/environments/contrail/ips-from-pool-all.yaml
```

2. Provide subscription manager credentials.

Use the template `environment-rhel-registration.yaml` to provide subscription manager credentials, including `rhel_reg_password`, `rhel_reg_pool_id`, `rhel_reg_repos`, `rhel_reg_user`, and `method`.

```
vi ~/tripleo-heat-templates/extraconfig/pre_deploy/
rhel-registration/environment-rhel-registration.yaml
```

The following is a sample `environment-rhel-registration.yaml` file for deployment.



NOTE: The repos enabled are required to enable deployment for Contrail 3.2 with Director 10 and OpenStack Newton.

```
[stack@instack ~]$ cat environment-rhel-registration.yaml
# Note this can be specified either in the call
# to heat stack-create via an additional -e option
# or via the global environment on the seed in
# /etc/heat/environment.d/default.yaml
parameter_defaults:
  rhel_reg_activation_key: ""
  rhel_reg_auto_attach: "true"
  rhel_reg_base_url: ""
  rhel_reg_environment: ""
  rhel_reg_force: ""
  rhel_reg_machine_name: ""
  rhel_reg_org: ""
  rhel_reg_password: ""
  rhel_reg_pool_id: ""
  rhel_reg_release: ""
  rhel_reg_repos: "rhel-7-server-rpms rhel-7-server-extras-rpms
rhel-7-server-rh-common-rpms rhel-ha-for-rhel-7-server-rpms
rhel-7-server-openstack-10-rpms rhel-7-server-openstack-10-devtools-rpms"
  rhel_reg_sat_url: ""
  rhel_reg_server_url: ""
  rhel_reg_service_level: ""
  rhel_reg_user: ""
  rhel_reg_type: ""
  rhel_reg_method: "portal"
  rhel_reg_sat_repo: "rhel-7-server-satellite-tools-6.1-rpms"
```

3. Set the overcloud nameserver.

```
neutron subnet-show neutron subnet-update <SUBNET-UUID> --dns-nameserver
NAMESERVER_IP
```

Deploying the Overcloud

- [Deploy Overcloud with a Single NIC on page 147](#)
- [Deploy Overcloud with Multiple NICs on page 147](#)
- [Deploy Overcloud with Multiple NICs with Bond and VLAN on page 147](#)

When you perform the overcloud installation, the overcloud is generated with the definitions you provide in the Heat templates.

The **openstack overcloud deploy** command creates a nested stack with all the resources needed to deploy the overcloud roles, networks, services, and so on.

- The stack can be updated if you wish to make changes to the overcloud.
- To redeploy the overcloud with a fresh installation, you delete the existing stack, make appropriate changes to the Heat templates, and then redeploy the stack.

Deploy Overcloud with a Single NIC

```
openstack overcloud deploy --templates tripleo-heat-templates/ \
  --roles-file tripleo-heat-templates/environments/contrail/roles_data.yaml \
  -e tripleo-heat-templates/extraconfig/pre_deploy/rhel-registration/
environment-rhel-registration.yaml \
  -e tripleo-heat-templates/extraconfig/pre_deploy/rhel-registration/
rhel-registration-resource-registry.yaml \
  -e tripleo-heat-templates/environments/contrail/contrail-services.yaml \
  -e tripleo-heat-templates/environments/contrail/contrail-net-single.yaml
```

Deploy Overcloud with Multiple NICs

```
openstack overcloud deploy --templates tripleo-heat-templates/ \
  --roles-file tripleo-heat-templates/environments/contrail/roles_data.yaml \
  -e tripleo-heat-templates/environments/puppet-pacemaker.yaml \
  -e tripleo-heat-templates/environments/contrail/contrail-services.yaml \
  -e tripleo-heat-templates/environments/contrail/network-isolation.yaml \
  -e tripleo-heat-templates/environments/contrail/contrail-net.yaml \
  -e tripleo-heat-templates/environments/contrail/ips-from-pool-all.yaml \
  -e tripleo-heat-templates/environments/network-management.yaml \
  -e tripleo-heat-templates/extraconfig/pre_deploy/rhel-registration/
environment-rhel-registration.yaml \
  -e tripleo-heat-templates/extraconfig/pre_deploy/rhel-registration/
rhel-registration-resource-registry.yaml
```

Deploy Overcloud with Multiple NICs with Bond and VLAN

```
openstack overcloud deploy --templates tripleo-heat-templates/ \
  --roles-file tripleo-heat-templates/environments/contrail/roles_data.yaml \
  -e tripleo-heat-templates/environments/puppet-pacemaker.yaml \
  -e tripleo-heat-templates/environments/contrail/contrail-services.yaml \
  -e tripleo-heat-templates/environments/contrail/network-isolation.yaml \
  -e tripleo-heat-templates/environments/contrail/contrail-net-bond-vlan.yaml \
  -e tripleo-heat-templates/environments/contrail/ips-from-pool-all.yaml \
  -e tripleo-heat-templates/environments/network-management.yaml \
```

```
-e tripleo-heat-templates/extraconfig/pre_deploy/rhel-registration/
environment-rhel-registration.yaml \
-e tripleo-heat-templates/extraconfig/pre_deploy/rhel-registration/
rhel-registration-resource-registry.yaml
```

Sample instackenv.json

This section has a sample **instackenv.json**, with OpenStack and Contrail controller on separate physical machines. This sample imports VMs to Ironic.

The sample **instackenv.json** is from a working environment that includes:

- 3x KVM hosts: 10.xx.xx.22, 10.xx.xx.24, 10.xx.xx.25 2.
- The following overcloud VMs on each KVM host:
 - openstack-controller
 - contrail-controller
 - contrail-analytics
 - contrail-analytics database
 - compute
- This sample imports VMs to Ironic.

Mandatory parameters for importing VMs to Ironic include:

Pm_addr—the IP address of the host on which the target VM is spawned.

Pm_user—Preferably the root user, or any other user with required permissions for accessing libvirtd.

Pm_password—the public SSH key of the host on which the target VM is spawned. Make sure that the line breaks are replaced with '\n'. You can use a simple program such as 'awk '{printf "%s\\n", \$0}' ~/.ssh/id_rsa' to achieve this.

MAC—the MAC address of the target VM's NIC that is connected to the provisioning control plane network.

Pm_type—pxe_ssh, the driver needed to provision VMs.

Sample instackenv.json

```
{
  "arch": "x86_64",
  "host-ip": "192.168.122.1",
  "power_manager": "nova.virt.baremetal.virtual_power_driver.VirtualPowerManager",

  "seed-ip": "",
  "ssh-key": "-----BEGIN RSA PRIVATE KEY-----
    $ABC123
  -----END RSA PRIVATE KEY-----\n",
  "ssh-user": "root",
  "nodes": [
    {
      "mac": [
```

```

        "52:54:00:d7:e4:87"
    ],
    "name": "control_1_at_5b5s36",
    "capabilities": "profile:control",
    "cpu": "4",
    "memory": "16384",
    "disk": "50",
    "arch": "x86_64",
    "pm_user": "root",
    "pm_addr": "10.xx.xx.24",
    "pm_password": "-----BEGIN RSA PRIVATE KEY-----
$ABC123
-----END RSA PRIVATE KEY-----",
    "pm_type": "pxe_ssh"
  }
},
{
  "mac": [
    "52:54:00:82:0d:9e"
  ],
  "name": "compute_1_at_5b5s36",
  "capabilities": "profile:compute",
  "cpu": "4",
  "memory": "16384",
  "disk": "50",
  "arch": "x86_64",
  "pm_user": "root",
  "pm_addr": "10.xx.xx.24",
  "pm_password": "-----BEGIN RSA PRIVATE KEY-----
$ABC123
-----END RSA PRIVATE KEY-----",
  "pm_type": "pxe_ssh"
}
},
{
  "mac": [
    "52:54:00:a2:ff:7a"
  ],
  "name": "contrail-controller_1_at_5b5s36",
  "capabilities": "profile:contrail-controller",
  "cpu": "4",
  "memory": "16384",
  "disk": "50",
  "arch": "x86_64",
  "pm_user": "root",
  "pm_addr": "10.xx.xx.24",
  "pm_password": "-----BEGIN RSA PRIVATE KEY-----
$ABC123
-----END RSA PRIVATE KEY-----",
  "pm_type": "pxe_ssh"
}
},
{
  "mac": [
    "52:54:00:51:35:bd"
  ],
  "name": "contrail-analytics_1_at_5b5s36",
  "capabilities": "profile:contrail-analytics",
  "cpu": "4",
  "memory": "16384",
  "disk": "50",

```

```

        "arch": "x86_64",
        "pm_user": "root",
        "pm_addr": "10.xx.xx.24",
        "pm_password": "-----BEGIN RSA PRIVATE KEY-----
$ABC123
-----END RSA PRIVATE KEY-----",
        "pm_type": "pxe_ssh"
    }
},
{
    "mac": [
        "52:54:00:a1:ae:4d"
    ],
    "name": "contrail-analytics-database_1_at_5b5s36",
    "capabilities": "profile:contrail-analytics-database",
    "cpu": "4",
    "memory": "16384",
    "disk": "50",
    "arch": "x86_64",
    "pm_user": "root",
    "pm_addr": "10.xx.xx.24",
    "pm_password": "-----BEGIN RSA PRIVATE KEY-----
$ABC123
-----END RSA PRIVATE KEY-----",
    "pm_type": "pxe_ssh"
}
},
{
    "mac": [
        "52:54:00:8b:0e:b8"
    ],
    "name": "control_1_at_5b5s34",
    "capabilities": "profile:control",
    "cpu": "4",
    "memory": "16384",
    "disk": "50",
    "arch": "x86_64",
    "pm_user": "root",
    "pm_addr": "10.xx.xx.22",
    "pm_password": "-----BEGIN RSA PRIVATE KEY-----
$ABC123
-----END RSA PRIVATE KEY-----",
    "pm_type": "pxe_ssh"
}
},
{
    "mac": [
        "52:54:00:c5:ba:b0"
    ],
    "name": "compute_1_at_5b5s34",
    "capabilities": "profile:compute",
    "cpu": "4",
    "memory": "16384",
    "disk": "50",
    "arch": "x86_64",
    "pm_user": "root",
    "pm_addr": "10.xx.xx.22",
    "pm_password": "-----BEGIN RSA PRIVATE KEY-----
$ABC123
-----END RSA PRIVATE KEY-----",
    "pm_type": "pxe_ssh"
}

```

```

    }
  ,
  {
    "mac": [
      "52:54:00:b8:5b:aa"
    ],
    "name": "contrail-controller_1_at_5b5s34",
    "capabilities": "profile:contrail-controller",
    "cpu": "4",
    "memory": "16384",
    "disk": "50",
    "arch": "x86_64",
    "pm_user": "root",
    "pm_addr": "10.xx.xx.22",
    "pm_password": "-----BEGIN RSA PRIVATE KEY-----
$ABC123
-----END RSA PRIVATE KEY-----",
    "pm_type": "pxe_ssh"
  }
  ,
  {
    "mac": [
      "52:54:00:2a:38:f1"
    ],
    "name": "contrail-analytics_1_at_5b5s34",
    "capabilities": "profile:contrail-analytics",
    "cpu": "4",
    "memory": "16384",
    "disk": "50",
    "arch": "x86_64",
    "pm_user": "root",
    "pm_addr": "10.xx.xx.22",
    "pm_password": "-----BEGIN RSA PRIVATE KEY-----
$ABC123
-----END RSA PRIVATE KEY-----",
    "pm_type": "pxe_ssh"
  }
  ,
  {
    "mac": [
      "52:54:00:fc:b7:67"
    ],
    "name": "contrail-analytics-database_1_at_5b5s34",
    "capabilities": "profile:contrail-analytics-database",
    "cpu": "4",
    "memory": "16384",
    "disk": "50",
    "arch": "x86_64",
    "pm_user": "root",
    "pm_addr": "10.xx.xx.22",
    "pm_password": "-----BEGIN RSA PRIVATE KEY-----
$ABC123
-----END RSA PRIVATE KEY-----",
    "pm_type": "pxe_ssh"
  }
  ,
  {
    "mac": [
      "52:54:00:48:b0:9b"
    ],
    "name": "control_1_at_5b5s37",

```

```

        "capabilities" : "profile:control",
        "cpu": "4",
        "memory": "16384",
        "disk": "50",
        "arch": "x86_64",
        "pm_user": "root",
        "pm_addr": "10.xx.xx.25",
        "pm_password": "-----BEGIN RSA PRIVATE KEY-----
$ABC123
-----END RSA PRIVATE KEY-----",
        "pm_type": "pxe_ssh"
    }
    ,
    {
        "mac": [
            "52:54:00:b3:01:b8"
        ],
        "name": "compute_1_at_5b5s37",
        "capabilities" : "profile:compute",
        "cpu": "4",
        "memory": "16384",
        "disk": "50",
        "arch": "x86_64",
        "pm_user": "root",
        "pm_addr": "10.xx.xx.25",
        "pm_password": "-----BEGIN RSA PRIVATE KEY-----
$ABC123
-----END RSA PRIVATE KEY-----",
        "pm_type": "pxe_ssh"
    }
    ,
    {
        "mac": [
            "52:54:00:9a:8c:f8"
        ],
        "name": "contrail-controller_1_at_5b5s37",
        "capabilities" : "profile:contrail-controller",
        "cpu": "4",
        "memory": "16384",
        "disk": "50",
        "arch": "x86_64",
        "pm_user": "root",
        "pm_addr": "10.xx.xx.25",
        "pm_password": "-----BEGIN RSA PRIVATE KEY-----
$ABC123
-----END RSA PRIVATE KEY-----",
        "pm_type": "pxe_ssh"
    }
    ,
    {
        "mac": [
            "52:54:00:8d:3d:d9"
        ],
        "name": "contrail-analytics_1_at_5b5s37",
        "capabilities" : "profile:contrail-analytics",
        "cpu": "4",
        "memory": "16384",
        "disk": "50",
        "arch": "x86_64",
        "pm_user": "root",
        "pm_addr": "10.xx.xx.25",
    }

```



```

        "pm_password": "-----BEGIN RSA PRIVATE KEY-----
$ABC123
-----END RSA PRIVATE KEY-----",
        "pm_type": "pxe_ssh"
    }
},
{
    "mac": [
        "52:54:00:9d:9e:57"
    ],
    "name": "contrail-analytics-database_1_at_5b5s37",
    "capabilities": "profile:contrail-analytics-database",
    "cpu": "4",
    "memory": "16384",
    "disk": "50",
    "arch": "x86_64",
    "pm_user": "root",
    "pm_addr": "10.xx.xx.25",
    "pm_password": "-----BEGIN RSA PRIVATE KEY-----
$ABC123
-----END RSA PRIVATE KEY-----",
    "pm_type": "pxe_ssh"
}
]
}

```

Adding a New Physical Compute Node

The following is a sample instackenv.json for adding a new physical compute node, by importing the physical compute or bare metal server to Ironic.

```

{
  "nodes": [
    {
      "mac": [
        "00:1b:21:99:ce:94"
      ],
      "name": "physical-compute_5b5s35",
      "capabilities": "profile:compute",
      "pm_user": "ADMIN",
      "pm_addr": "10.xx.xxx.206",
      "pm_password": "ADMIN",
      "pm_type": "pxe_ipmitool"
    }
  ]
}

```

The following are the mandatory parameters to import a physical compute or bare metal server to Ironic.

Pm_addr—Server's IPMI

Pm_user—IPMI user name

Pm_password— IPMI password

MAC—MAC address of the server's NIC that is connected to the provisioning/control-plane network

Pm_type—pxe_ipmitoo

Specify this driver to provision physical servers.

Requirements for Deploying to VMs

The following are required for deploying to VMs.

- Password-less SSH must be set up from the undercloud to all servers that will host overcloud VMs, for the user 'root'.
- Libvirt on KVM hosts must be configured to allow TCP sessions without requiring Transport Layer Security (TLS).

Sample Heat Templates for NICs

This section provides sample Heat templates for different configurations for NICs.

- [Example 1: NIC-1 to control plane; NIC-2 bridged interface on page 154](#)
- [Example 2: NIC-1 to control plane; NIC-2 and NIC-3 bond interface; NIC-4 other networks on page 159](#)

Example 1: NIC-1 to control plane; NIC-2 bridged interface

This sample has the following topology:

- NIC-1 is connected to the control plane provisioning network
Connected to an access port on the underlay switch
- NIC-2 is a bridged interface, and has a unique VLAN tag for each of the other overlay networks.

Underlying switch configuration:

- NIC-1 is connected to the control plane provisioning VLAN access-ports of a switch.
- NIC-2 is connected to trunk ports on the switch. The trunk ports will carry multiple VLAN tags, one each for the following networks:

VLAN-10: External VLAN

VLAN-20: Internal API VLAN

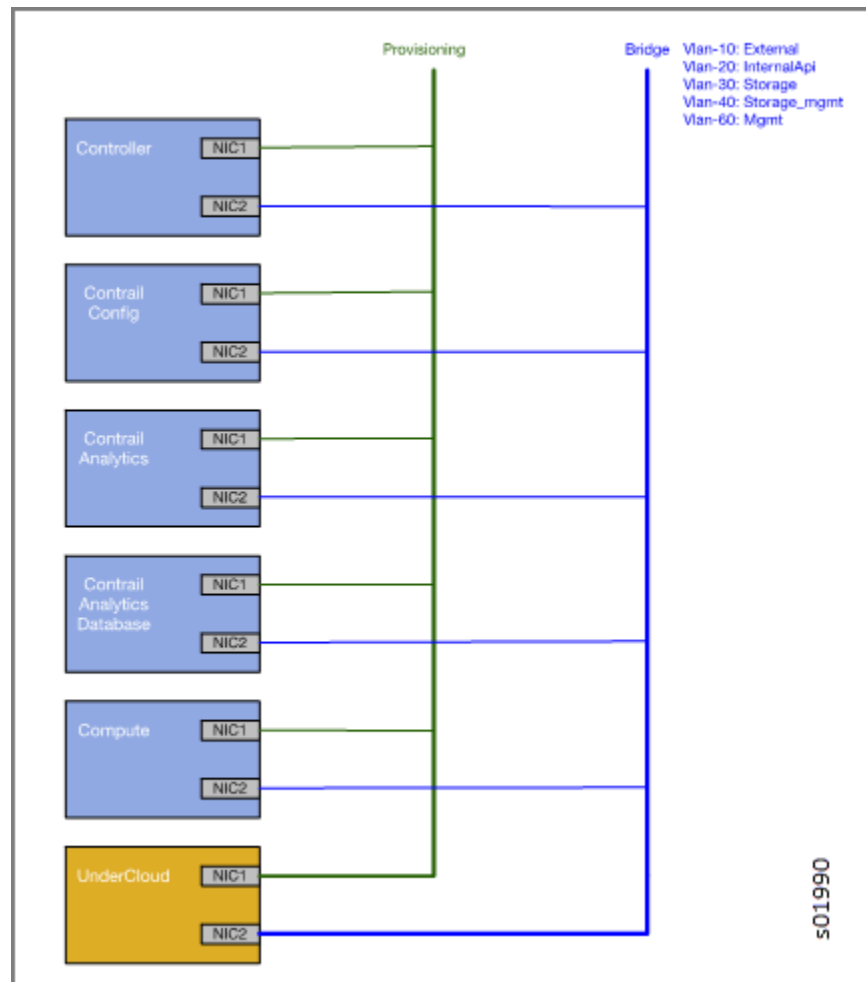
VLAN-30: Storage VLAN

VLAN-40: Storage-MGMT VLAN

VLAN-60: Management VLAN

[Figure 31 on page 155](#) shows the server NIC configuration for this example.

Figure 31: Server NIC Configuration



NIC Template

The following is the NIC template to configure the setup in this example.

Note: For this setup, the default route is reachable by means of the InternalAPI network.

```
heat_template_version: 2015-04-30
```

```
description: >
  Software Config to drive os-net-config to configure multiple interfaces
  for the compute role.
```

```
parameters:
  ControlPlaneIp:
    default: ''
    description: IP address/subnet on the ctlplane network
    type: string
  ExternalIpSubnet:
    default: ''
    description: IP address/subnet on the external network
```

```

    type: string
InternalApiIpSubnet:
  default: ''
  description: IP address/subnet on the internal API network
  type: string
InternalApiDefaultRoute: # Not used by default in this template
  default: '10.0.0.1'
  description: The default route of the internal api network.
  type: string
StorageIpSubnet:
  default: ''
  description: IP address/subnet on the storage network
  type: string
StorageMgmtIpSubnet:
  default: ''
  description: IP address/subnet on the storage mgmt network
  type: string
TenantIpSubnet:
  default: ''
  description: IP address/subnet on the tenant network
  type: string
ManagementIpSubnet: # Only populated when including
environments/network-management.yaml
  default: ''
  description: IP address/subnet on the management network
  type: string
ExternalNetworkVlanID:
  default: 10
  description: Vlan ID for the external network traffic.
  type: number
InternalApiNetworkVlanID:
  default: 20
  description: Vlan ID for the internal_api network traffic.
  type: number
StorageNetworkVlanID:
  default: 30
  description: Vlan ID for the storage network traffic.
  type: number
StorageMgmtNetworkVlanID:
  default: 40
  description: Vlan ID for the storage mgmt network traffic.
  type: number
TenantNetworkVlanID:
  default: 50
  description: Vlan ID for the tenant network traffic.
  type: number
ManagementNetworkVlanID:
  default: 60
  description: Vlan ID for the management network traffic.
  type: number
ControlPlaneSubnetCidr: # Override this via parameter_defaults
  default: '24'
  description: The subnet CIDR of the control plane network.
  type: string
ControlPlaneDefaultRoute: # Override this via parameter_defaults
  description: The default route of the control plane network.
  type: string
ExternalInterfaceDefaultRoute: # Not used by default in this template
  default: '10.0.0.1'
  description: The default route of the external network.
  type: string

```

```

ManagementInterfaceDefaultRoute: # Commented out by default in this template
  default: unset
  description: The default route of the management network.
  type: string
DnsServers: # Override this via parameter_defaults
  default: []
  description: A list of DNS servers (2 max for some implementations) that will
be added to resolv.conf.
  type: comma_delimited_list
EC2MetadataIp: # Override this via parameter_defaults
  description: The IP address of the EC2 metadata server.
  type: string
resources:
  OsNetConfigImpl:
    type: OS::Heat::StructuredConfig
    properties:
      group: os-apply-config
      config:
        os_net_config:
          network_config:
            -
              type: interface
              name: nic1
              use_dhcp: false
              dns_servers: {get_param: DnsServers}
              addresses:
                -
                  ip_netmask:
                    list_join:
                      - '/'
                      - - {get_param: ControlPlaneIp}
                        - {get_param: ControlPlaneSubnetCidr}
              routes:
                -
                  ip_netmask: 1xx.254.1xx.254/32
                  next_hop: {get_param: EC2MetadataIp}
            -
              type: vlan
              use_dhcp: false
              vlan_id: {get_param: InternalApiNetworkVlanID}
              device: nic2
              addresses:
                -
                  ip_netmask: {get_param: InternalApiIpSubnet}
              routes:
                -
                  default: true
                  next_hop: {get_param: InternalApiDefaultRoute}
            -
              type: vlan
              vlan_id: {get_param: ManagementNetworkVlanID}
              device: nic2
              addresses:
                -
                  ip_netmask: {get_param: ManagementIpSubnet}
            -
              type: vlan
              vlan_id: {get_param: ExternalNetworkVlanID}
              device: nic2
              addresses:
                -

```

```

        ip_netmask: {get_param: ExternalIpSubnet}
-
    type: vlan
    vlan_id: {get_param: StorageNetworkVlanID}
    device: nic2
    addresses:
    -
        ip_netmask: {get_param: StorageIpSubnet}
-
    type: vlan
    vlan_id: {get_param: StorageMgmtNetworkVlanID}
    device: nic2
    addresses:
    -
        ip_netmask: {get_param: StorageMgmtIpSubnet}

outputs:
  OS::stack_id:
    description: The OsNetConfigImpl resource.
    value: {get_resource: OsNetConfigImpl}

```

NIC definitions of the corresponding compute file are the following.

```

network_config:
-
    type: interface
    name: nic1
    use_dhcp: false
    dns_servers: {get_param: DnsServers}
    addresses:
    -
        ip_netmask:
            list_join:
            - '/'
            - {get_param: ControlPlaneIp}
            - {get_param: ControlPlaneSubnetCidr}
    routes:
    -
        ip_netmask: 169.xxx.xxx.254/32
        next_hop: {get_param: EC2MetadataIp}
-
    type: interface
    name: vhost0
    use_dhcp: false
    addresses:
    -
        ip_netmask: {get_param: InternalApiIpSubnet}
    routes:
    -
        default: true
        next_hop: {get_param: InternalApiDefaultRoute}
-
    type: vlan
    vlan_id: {get_param: ExternalNetworkVlanID}
    device: nic2
    addresses:
    -
        ip_netmask: {get_param: ExternalIpSubnet}
-
    type: vlan

```

```

      vlan_id: {get_param: StorageNetworkVlanID}
      device: nic2
      addresses:
      -
        ip_netmask: {get_param: StorageIpSubnet}
-
  type: vlan
  vlan_id: {get_param: StorageMgmtNetworkVlanID}
  device: nic2
  addresses:
  -
    ip_netmask: {get_param: StorageMgmtIpSubnet}

```

Example 2: NIC-1 to control plane; NIC-2 and NIC-3 bond interface; NIC-4 other networks

This sample has the following topology:

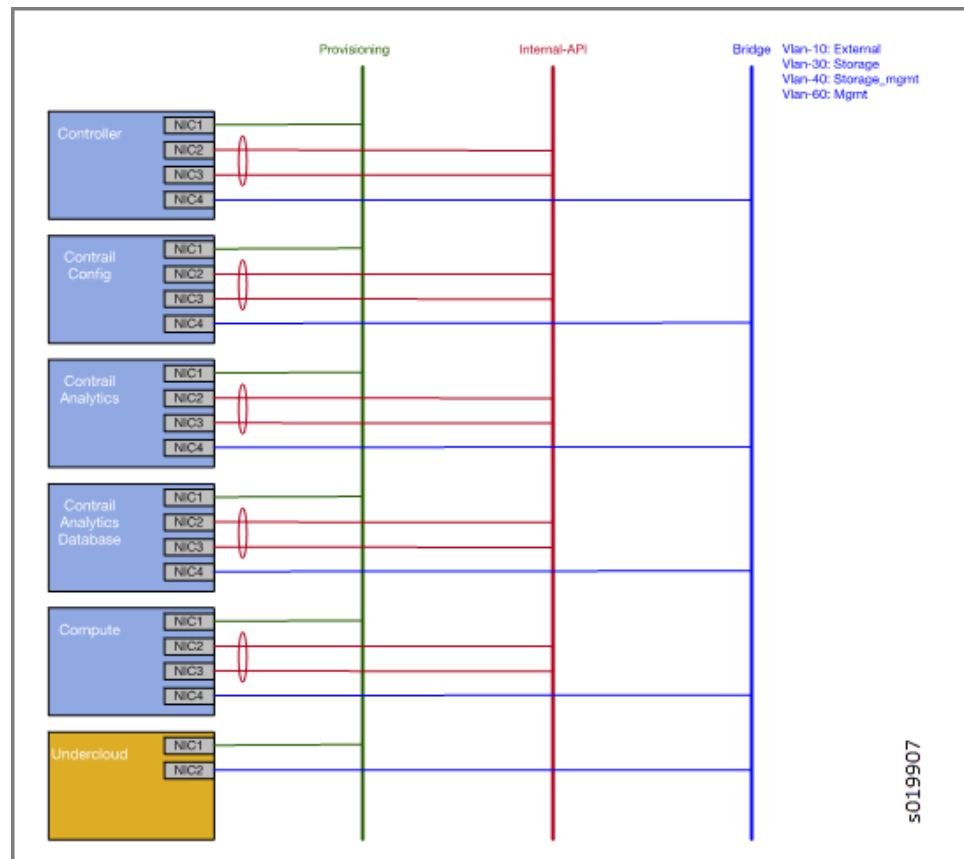
- NIC-1 is connected to the control plane provisioning network
Connected to an access port on the underlay switch
- NIC-2 and NIC-3 are connected to the InternalAPI network.
These two NICs are part of a bond interface.
- NIC-4 has a unique VLAN tag for each of the other overlay networks. It carries the rest of the networks.

Underlying switch configuration:

- NIC-1 is connected to the control plane provisioning VLAN access-ports of a switch.
- NIC-2 and NIC-3 connected to access ports on the switch in the InternalAPI VLAN.
These switch ports are bundled together as a LAG
- NIC-4 is connected to trunk ports on the switch. The trunk ports will carry multiple VLAN tags, one each for the following networks:
VLAN-10: External VLAN
VLAN-30: Storage VLAN
VLAN-40: Storage-MGMT VLAN
VLAN-60: Management VLAN

[Figure 32 on page 160](#) shows the server NIC configuration for this example.

Figure 32: Server NIC Configuration



NIC Template

The following is a snippet of the corresponding NIC template to configure the setup in this example.

Note: For this setup, the default route is reachable by means of the InternalAPI network.

```
resources:
  OsNetConfigImpl:
    type: OS::Heat::StructuredConfig
    properties:
      group: os-apply-config
      config:
        os_net_config:
          network_config:
            -
              type: interface
              name: nic1
              use_dhcp: false
              dns_servers: {get_param: DnsServers}
              addresses:
                -
                  ip_netmask:
                    list_join:
                      - '/'
```



```

        - - {get_param: ControlPlaneIp}
        - {get_param: ControlPlaneSubnetCidr}
    routes:
    -
        ip_netmask: 169.xxx.xxx.254/32
        next_hop: {get_param: EC2MetadataIp}
-
    type: linux_bond
    name: bond0
    use_dhcp: false
    addresses:
    -
        ip_netmask: {get_param: InternalApiIpSubnet}
    routes:
    -
        default: true
        next_hop: {get_param: InternalApiDefaultRoute}
    bonding_options: "mode=active-active"
    members:
    -
        type: interface
        name: nic2
    -
        type: interface
        name: nic3
-
    type: vlan
    vlan_id: {get_param: ManagementNetworkVlanID}
    device: nic4
    addresses:
    -
        ip_netmask: {get_param: ManagementIpSubnet}
-
    type: vlan
    vlan_id: {get_param: ExternalNetworkVlanID}
    device: nic4
    addresses:
    -
        ip_netmask: {get_param: ExternalIpSubnet}
-
    type: vlan
    vlan_id: {get_param: StorageNetworkVlanID}
    device: nic4
    addresses:
    -
        ip_netmask: {get_param: StorageIpSubnet}
-
    type: vlan
    vlan_id: {get_param: StorageMgmtNetworkVlanID}
    device: nic4
    addresses:
    -
        ip_netmask: {get_param: StorageMgmtIpSubnet}

outputs:
  OS::stack_id:
    description: The OsNetConfigImpl resource.
    value: {get_resource: OsNetConfigImpl}

```

More Template Examples

More template examples are available in the directory:

`/home/stack/tripleo-heat-templates/environments/contrail`

There are separate templates for control-plane and compute. You can modify the example templates to match your topology.

```
[stack@instack contrail]$ pwd
/home/stack/tripleo-heat-templates/environments/contrail
[stack@instack contrail]$ ls -lrt | grep nic | grep compute
-rw-rw-r--. 1 stack stack 6136 May 31 15:07
contrail-nic-config-compute-bond-vlan.yaml
-rw-rw-r--. 1 stack stack 5839 May 31 15:07
contrail-nic-config-compute-bond-vlan-dpdk.yaml
-rw-rw-r--. 1 stack stack 5669 May 31 15:07
contrail-nic-config-compute-storage-mgmt.yaml
-rw-rw-r--. 1 stack stack 3864 May 31 15:07
contrail-nic-config-compute-single.yaml
-rw-rw-r--. 1 stack stack 5422 May 31 15:07 contrail-nic-config-compute-dpdk.yaml
-rw-rw-r--. 1 stack stack 5643 Jun 1 11:56
contrail-nic-config-compute-dpdk-bond-vlan.yaml
-rw-rw-r--. 1 stack stack 5661 Jun 2 12:43 contrail-nic-config-compute.yaml
[stack@instack contrail]$
[stack@instack contrail]$
[stack@instack contrail]$ ls -lrt | grep nic | grep -v compute
-rw-rw-r--. 1 stack stack 5568 May 31 15:07 contrail-nic-config-storage-mgmt.yaml
-rw-rw-r--. 1 stack stack 3861 May 31 15:07 contrail-nic-config-single.yaml
-rw-rw-r--. 1 stack stack 6688 May 31 15:07 contrail-nic-config-ovs-bond.yaml
-rw-rw-r--. 1 stack stack 5793 Jun 1 11:46 contrail-nic-config-vlan.yaml
-rw-rw-r--. 1 stack stack 5793 Jun 2 11:54 contrail-nic-config.yaml
```

What are NIC Templates?

TripleO (OpenStack On OpenStack) provides the flexibility to have different NIC templates for different overcloud roles. For example, there might be differences between the NIC and networking layout for the **overcloud-compute-nodes** and the **overcloud-contrail-controller-nodes**.

How NIC Templates Work

The NIC templates provide data to the backend scripts that take care of provisioning the network on the overcloud nodes. The templates are written in standard JSON formats.

The **resources** section within each template contains all of the networking information for the corresponding overcloud role, including:

- Number of NICs
- Network associated with each NIC
- Static routes associated with each NIC
- Any VLAN configuration which is tied to a particular NIC
 - Network associated with each VLAN interface

- Static routes associated with each VLAN

For more information on what each of these sections looks like, see Red Hat documentation: https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/10/html/advanced_overcloud_customization/sect-isolating_networks

The Red Hat documentation has many examples of how to define a NIC within the template, and some of that information is used in the examples in this topic.

A limitation in Red Hat Director 10 is that all of the overcloud networks must be stretched at Layer 2 to all of the overcloud nodes. If the overcloud nodes are physical servers that are present in different racks or subnets of an IP fabric, then you'll have to first stretch all the overcloud networks to the physical servers. One way to do this is to use EVPN. If you have a traditional datacenter topology (non-IP fabric), then you can extend VLANs across the physical computes to extend all the overcloud networks.

Deploying an overcloud using TripleO and Director across multiple subnets is an upstream feature and a work-in-progress at this time. Upstream developers (mostly from Red Hat) are driving this effort. To check the status of this feature, see: <https://blueprints.launchpad.net/tripleo/+spec/tripleo-routed-networks-deployment>

Common Topologies

One of the most common topologies for a TripleO deployment consists of 3 NICs:

- NIC-1: Carries these networks:
 - Provisioning: Untagged
 - Management: Tagged
 - External: Tagged
- NIC-2: Carries internal-API network
- NIC-3: Carries tagged storage related networks (storage and storage management)

Conventions in this Document

Examples are provided in this document.

- The topology used in the examples has the following constraints:
 - The first NIC must be connected to the ControlPlane network.
 - The second NIC must have separate VLAN interfaces for every other network.
- With the above limitations, 'eth1' is specified as the VlanParentInterface.
- Note that 'nic-2' is specified as the interface with multiple VLAN sub-interfaces in the NIC definition template.
- In the current version of RHEL 7.3/7.4, the NICs manifest as eth0, eth1, and so on. Because of this, NIC-2 translates to eth-1.

There are several NIC templates within Contrail that are available to users. These templates are named according to the topology that they're trying to solve, and are available in the **environments/contrail/** directory. Please modify these templates according to your topology before deploying Contrail with TripleO/Red Hat Director.

Contrail NIC Templates

As part of deployment, a network (net) template must be provided. The net template files are all available at the same location:

Sample Net Templates

```
[stack@undercloud contrail]$ ls -lrt | grep contrail-net
-rw-rw-r--. 1 stack stack 1866 Sep 19 17:10 contrail-net-storage-mgmt.yaml
-rw-rw-r--. 1 stack stack  894 Sep 19 17:10 contrail-net-single.yaml
-rw-rw-r--. 1 stack stack 1528 Sep 19 17:10 contrail-net-dpdk.yaml
-rw-rw-r--. 1 stack stack 1504 Sep 19 17:10 contrail-net-bond-vlan.yaml
-rw-rw-r--. 1 stack stack 1450 Sep 19 17:12 contrail-net.yaml
```

The template files are prepopulated examples that are included with a Contrail package. The file names match the use case that each is trying to solve. For example, use the **contrail-net-dpdk.yaml** file if your use case includes a DPDK compute. Similarly, use the **contrail-net-bond-vlan.yaml** file if your topology uses bond interfaces and VLAN subinterfaces that need to be created on top of the bond interfaces.

Please note that these are example files, and you'll need to modify them to match your topology.

Resource Registry Example

The **resource_registry** section of the net template file specifies which NIC template must be used for each role:

Sample Resource Registry of Net Template

```
[stack@undercloud contrail]$ cat contrail-net.yaml
resource_registry:
  OS::TripleO::Compute::Net::SoftwareConfig: contrail-nic-config-compute.yaml
  OS::TripleO::ContrailDpdk::Net::SoftwareConfig:
    contrail-nic-config-compute-dpdk-bond-vlan.yaml
  OS::TripleO::Controller::Net::SoftwareConfig: contrail-nic-config.yaml
  OS::TripleO::ContrailController::Net::SoftwareConfig: contrail-nic-config.yaml

  OS::TripleO::ContrailAnalytics::Net::SoftwareConfig: contrail-nic-config.yaml
  OS::TripleO::ContrailAnalyticsDatabase::Net::SoftwareConfig:
    contrail-nic-config.yaml
  OS::TripleO::ContrailTsn::Net::SoftwareConfig: contrail-nic-config-compute.yaml

parameter_defaults:
  ControlPlaneSubnetCidr: '24'
  InternalApiNetCidr: 10.0.0.0/24
  InternalApiAllocationPools: [{'start': '10.0.0.10', 'end': '10.0.0.200'}]
  InternalApiDefaultRoute: 10.0.0.1
  ManagementNetCidr: 10.1.0.0/24
  ManagementAllocationPools: [{'start': '10.1.0.10', 'end': '10.1.0.200'}]
  ManagementInterfaceDefaultRoute: 10.1.0.1
  ExternalNetCidr: 10.2.0.0/24
```

```

ExternalAllocationPools: [{'start': '10.2.0.10', 'end': '10.2.0.200'}]
EC2MetadataIp: 192.0.2.1 # Generally the IP of the Undercloud
DnsServers: ["8.8.8.8", "8.8.4.4"]
VrouterPhysicalInterface: vlan20
VrouterGateway: 10.0.0.1
VrouterNetmask: 255.255.255.0
ControlVirtualInterface: eth0
PublicVirtualInterface: vlan10
VlanParentInterface: eth1 # If VrouterPhysicalInterface is a vlan interface
using vlanX notation

```

NIC Templates for Control Nodes

In this example, all of the OpenStack controller and the Contrail control plane roles use the NIC template named **contrail-nic-config.yaml**. Note that the compute roles and the DPDK roles use different NIC templates.

The NIC template files can be accessed at this location:

Sample NIC Templates

```

[stack@undercloud contrail]$ ls -lrt | grep contrail-nic-config-
-rw-rw-r--. 1 stack stack 5615 Sep 19 17:10 contrail-nic-config-vlan.yaml
-rw-rw-r--. 1 stack stack 5568 Sep 19 17:10 contrail-nic-config-storage-mgmt.yaml
-rw-rw-r--. 1 stack stack 3861 Sep 19 17:10 contrail-nic-config-single.yaml
-rw-rw-r--. 1 stack stack 5669 Sep 19 17:10
contrail-nic-config-compute-storage-mgmt.yaml
-rw-rw-r--. 1 stack stack 3864 Sep 19 17:10
contrail-nic-config-compute-single.yaml
-rw-rw-r--. 1 stack stack 5385 Sep 19 17:10 contrail-nic-config-compute-dpdk.yaml
-rw-rw-r--. 1 stack stack 5839 Sep 19 17:10
contrail-nic-config-compute-bond-vlan.yaml
-rw-rw-r--. 1 stack stack 5666 Sep 19 17:10
contrail-nic-config-compute-bond-vlan-dpdk.yaml
-rw-rw-r--. 1 stack stack 5538 Sep 19 17:10
contrail-nic-config-compute-bond-dpdk.yaml
-rw-rw-r--. 1 stack stack 5132 Sep 19 17:13 contrail-nic-config-compute.yaml
-rw-r--r--. 1 stack stack 5503 Sep 19 17:13
contrail-nic-config-compute-dpdk-bond-vlan.yaml

```

Just like the network template files, these NIC template files are examples which are included with the Contrail package. These files also have their names matching the use case that they're trying to solve.

Note that these NIC template files are examples, and you may have to modify these according to your cluster's topology.

Also, these examples call out NIC names in the format of `nic1`, `nic2`, `nic3`, and so on (`nic.<number>`). Think of these as variables, and Director's backend scripts translate these NIC numbers into actual interface names based on the interface boot order. So if you specify `nic1`, `nic2`, and `nic3` in the template and the boot order of interfaces is `eth0`, `eth1`, and `eth2`, then the mapping of these `nic` variables to actual interfaces would look like:

- `Nic1` mapped to `eth0`
- `Nic2` mapped to `eth1`

- Nic3 mapped to eth2

TripleO also provides the flexibility to use actual NIC names (eth0, em1, ens2f, and so on) in the NIC templates instead of using nic1, nic2, nic3, and the like.



NOTE: A common mistake while defining NIC templates is that the boot order of NICs is not set correctly. Because of this, your deployment might progress beyond the network configuration stage, but there might be connectivity issues because the IP/Subnet/route information might not be configured correctly for the NICs of overcloud nodes.

This section takes a zoom-in look at the `network_config` section of the NIC template used by the controllers: `contrail-nic-config.yaml`.

Sample Network Config for Control Nodes

```
network_config:
-
  type: interface
  name: nic1
  use_dhcp: false
  dns_servers: {get_param: DnsServers}
  addresses:
  -
    ip_netmask:
      list_join:
      - '/'
      - {get_param: ControlPlaneIp}
      - {get_param: ControlPlaneSubnetCidr}
  routes:
  -
    ip_netmask: 169.254.169.254/32
    next_hop: {get_param: EC2MetadataIp}
-
  type: vlan
  use_dhcp: false
  vlan_id: {get_param: InternalApiNetworkVlanID}
  device: nic2
  addresses:
  -
    ip_netmask: {get_param: InternalApiIpSubnet}
  routes:
  -
    default: true
    next_hop: {get_param: InternalApiDefaultRoute}
-
  type: vlan
  vlan_id: {get_param: ManagementNetworkVlanID}
  device: nic2
  addresses:
  -
    ip_netmask: {get_param: ManagementIpSubnet}
-
  type: vlan
  vlan_id: {get_param: ExternalNetworkVlanID}
  device: nic2
  addresses:
```

```

-
  ip_netmask: {get_param: ExternalIpSubnet}
-
  type: vlan
  vlan_id: {get_param: StorageNetworkVlanID}
  device: nic2
  addresses:
    -
      ip_netmask: {get_param: StorageIpSubnet}
-
  type: vlan
  vlan_id: {get_param: StorageMgmtNetworkVlanID}
  device: nic2
  addresses:
    -
      ip_netmask: {get_param: StorageMgmtIpSubnet}

```

NIC Control Node Template Subsection: Definition for NIC1

The subsection of the template for NIC1 includes the following.

- The definition for an interface called 'nic1'
- The DNS server is defined. Make sure that this parameter has a valid value. Most commonly, this variable is assigned a value in the **contrail-services.yaml** file.
- An IP and subnet is provided under the 'addresses' section. Note that these values are also variables, and the format is: **\$(Network_Name)IP** and **\$(Network_Name)SubnetCidr**.
 - This means that this particular NIC is on the ControlPlane network. In the background, this NIC might be connected to an access port on a switch for the ControlPlane VLAN.
- In the 'routes' section, there's a /32 route out of this NIC. At the time of planning the networking for your cluster, you may need to provision static routes on the overcloud roles. Use the format mentioned under the 'routes' section to specify any such static routes.

Sample NIC1

```

-
  type: interface
  name: nic1
  use_dhcp: false
  dns_servers: {get_param: DnsServers}
  addresses:
    -
      ip_netmask:
        list_join:
          - '/'
          - - {get_param: ControlPlaneIp}
            - {get_param: ControlPlaneSubnetCidr}
  routes:
    -
      ip_netmask: 169.254.169.254/32
      next_hop: {get_param: EC2MetadataIp}

```

NIC Template Subsection: Definition for NIC2

The subsection of the template for NIC2 includes the following.

- The NIC2 has multiple VLANs defined on it.
 - In the background, NIC2 might be connected to a switch's trunk port, and all of the corresponding VLANs must be allowed on the trunk.
 - Because Director-based deployments need the administrator to use a number of networks, it's a very common requirement or design to use VLAN interfaces on the overcloud nodes. Consequently, the administrators do not have to be concerned about having 6-7 physical NICs on each overcloud node.
- For each VLAN interface, the `vlan_id` is defined. Note that the `vlan_id` points to a variable. As with the example for NIC1, these variables can be assigned values in the **contrail-net.yaml**.
- Another important observation is the setting of the default route. In this example, the default route was provisioned on the VLAN interface in the InternalAPI network. Note that the next hop points to a variable. As with other variables, this variable can be set in the **contrail-net.yaml** file. The following snippet shows the default route information.

```
-
  type: vlan
  use_dhcp: false
  vlan_id: {get_param: InternalApiNetworkVlanID}
  device: nic2
  addresses:
    -
      ip_netmask: {get_param: InternalApiIpSubnet}
  routes:
    -
      default: true
      next_hop: {get_param: InternalApiDefaultRoute}
```

NIC Templates for Compute Nodes

The NIC definitions for compute roles are slightly different from the definitions for control nodes. This is because Contrail provisions a logical interface called 'vhost0' on all compute nodes, and this interface must be provided in the NIC definition file for a compute node. Vhost0 is the logical interface that gets attached to the control data network (or the InternalAPI network in TripleO-based installation).

In the `contrail-net.yaml` example provided in the beginning of this topic, the NIC template used for the compute nodes is **contrail-nic-config-compute.yaml**. The following is the 'resources' section of the **contrail-nic-config-compute.yaml** file:

Sample Resources for Compute Nodes

```
resources:
  OsNetConfigImpl:
    type: OS::Heat::StructuredConfig
    properties:
      group: os-apply-config
      config:
```



```

os_net_config:
  network_config:
    -
      type: interface
      name: nic1
      use_dhcp: false
      dns_servers: {get_param: DnsServers}
      addresses:
        -
          ip_netmask:
            list_join:
              - '/'
              - {get_param: ControlPlaneIp}
              - {get_param: ControlPlaneSubnetCidr}
      routes:
        -
          ip_netmask: 169.254.169.254/32
          next_hop: {get_param: EC2MetadataIp}
    -
      type: vlan
      vlan_id: {get_param: InternalApiNetworkVlanID}
      device: nic2
    -
      type: interface
      name: vhost0
      use_dhcp: false
      addresses:
        -
          ip_netmask: {get_param: InternalApiIpSubnet}
      routes:
        -
          default: true
          next_hop: {get_param: InternalApiDefaultRoute}
    -
      type: vlan
      vlan_id: {get_param: ManagementNetworkVlanID}
      device: nic2
      addresses:
        -
          ip_netmask: {get_param: ManagementIpSubnet}
    -
      type: vlan
      vlan_id: {get_param: ExternalNetworkVlanID}
      device: nic2
      addresses:
        -
          ip_netmask: {get_param: ExternalIpSubnet}
    -
      type: vlan
      vlan_id: {get_param: StorageNetworkVlanID}
      device: nic2
      addresses:
        -
          ip_netmask: {get_param: StorageIpSubnet}
    -
      type: vlan
      vlan_id: {get_param: StorageMgmtNetworkVlanID}
      device: nic2
      addresses:
        -
          ip_netmask: {get_param: StorageMgmtIpSubnet}

```

NIC Compute Node Template Subsection: Definition for NIC1

This section is very similar to the NIC1 definition template for the control nodes. In this example topology, the first NIC for all the compute nodes is connected to the ControlPlane network. Note that this is untagged, so this NIC might be connected to an access port on the underlay switch.

Sample NIC1 for Compute Node

```
-
  type: interface
  name: nic1
  use_dhcp: false
  dns_servers: {get_param: DnsServers}
  addresses:
  -
    ip_netmask:
      list_join:
        - '/'
        - - {get_param: ControlPlaneIp}
          - {get_param: ControlPlaneSubnetCidr}
  routes:
  -
    ip_netmask: 169.254.169.254/32
    next_hop: {get_param: EC2MetadataIp}
```

NIC Compute Node Template Subsection: Definition for NIC2

This section is very similar to the NIC2 definition template for the control nodes, however there are two major differences:

- The VLAN subinterface for **InternalApiNetwork** does not have an IP address.
- The Vhost0 interface holds the IP address for **InternalApiNetwork**.
 - If you're using stock TripleO-based installation, then the IP address for the **InternalApiNetwork** will always be configured on the **vhost0** interface.

Sample NIC2 for Compute Node

```
-
  type: interface
  name: vhost0
  use_dhcp: false
  addresses:
  -
    ip_netmask: {get_param: InternalApiIpSubnet}
  routes:
  -
    default: true
    next_hop: {get_param: InternalApiDefaultRoute}
-
  type: vlan
  vlan_id: {get_param: InternalApiNetworkVlanID}
  device: nic2
-
  type: vlan
  vlan_id: {get_param: ManagementNetworkVlanID}
```

```

device: nic2
addresses:
-
  ip_netmask: {get_param: ManagementIpSubnet}
-
  type: vlan
  vlan_id: {get_param: ExternalNetworkVlanID}
  device: nic2
  addresses:
  -
    ip_netmask: {get_param: ExternalIpSubnet}
-
  type: vlan
  vlan_id: {get_param: StorageNetworkVlanID}
  device: nic2
  addresses:
  -
    ip_netmask: {get_param: StorageIpSubnet}
-
  type: vlan
  vlan_id: {get_param: StorageMgmtNetworkVlanID}
  device: nic2
  addresses:
  -
    ip_netmask: {get_param: StorageMgmtIpSubnet}

```

The following are additional parameters that are required to successfully provision compute nodes. The parameters are handled as variables and are normally specified in the **contrail-net.yaml** file.

- **Network-related parameters:**
 - **Subnet CIDR:** You can set the subnet mask of each overcloud network in this file.
 - **Allocation Pool Range:** If set, then the overcloud nodes are allocated IP addresses from the specified range
 - **Default Route:** Set the next hop for the default route in the specified format. In this example, the default route is set for InternalApi network and the next hop is set as 10.0.0.1
- **VrouterPhysicalInterface:** This is the interface on which vhost0 interface gets attached. This may be a physical NIC (e.g. eth2 or enps0f0), or a VLAN interface (e.g. Vlan20)
- **VrouterGateway:** This is the IP address of the SDN gateway. In a lot of deployments, this might be the IP address of the MX router's IP address. This IP must be reachable via the InternalAPI network
- **VrouterNetmask:** subnet mask for the vhost0 interface (this is provisioned in the compute nodes' config files).
- **VlanParentInterface:** This is optional, and needed only if vhost0 needs to be attached to a VLAN interface.

Sample NIC2 Additional Parameters for Compute Node

```

parameter_defaults:
  ControlPlaneSubnetCidr: '24'

```

```
InternalApiNetCidr: 10.0.0.0/24
InternalApiAllocationPools: [{'start': '10.0.0.10', 'end': '10.0.0.200'}]
InternalApiDefaultRoute: 10.0.0.1
ManagementNetCidr: 10.1.0.0/24
ManagementAllocationPools: [{'start': '10.1.0.10', 'end': '10.1.0.200'}]
ManagementInterfaceDefaultRoute: 10.1.0.1
ExternalNetCidr: 10.2.0.0/24
ExternalAllocationPools: [{'start': '10.2.0.10', 'end': '10.2.0.200'}]
EC2MetadataIp: 192.0.2.1 # Generally the IP of the Undercloud
DnsServers: ["8.8.8.8","8.8.4.4"]
VrouterPhysicalInterface: vlan20
VrouterGateway: 10.0.0.1
VrouterNetmask: 255.255.255.0
ControlVirtualInterface: eth0
PublicVirtualInterface: vlan10
```

CHAPTER 8

Using Server Manager to Automate Provisioning

- [Installing Server Manager on page 173](#)
- [Using Server Manager to Automate Provisioning on page 178](#)
- [Using the Server Manager Web User Interface on page 206](#)
- [Installing and Using Server Manager Lite on page 225](#)

Installing Server Manager

- [Installation Requirements for Server Manager on page 173](#)
- [Installing Server Manager on page 174](#)
- [Upgrading Server Manager Software on page 176](#)
- [Server Manager Installation Completion Checks on page 176](#)
- [Sample Configurations for Server Manager Templates on page 177](#)

Installation Requirements for Server Manager

This document provides details for installing Server Manager.

Platform Support

As of Contrail 4.0, Server Manager can be installed on, and used to reimage and provision, the following platform operating systems:

- Ubuntu 16.04.01
- Ubuntu 16.04.02
- Ubuntu 14.04.5
- Ubuntu 14.04.4
- Ubuntu 14.04.2
- Ubuntu 14.04.1
- Ubuntu 14.04

As of Contrail 4.0, Server Manager installation supports Contrail provisioning for only the following OpenStack versions:

- Ocata, on Ubuntu 16.04 platform, only
- Newton, on Ubuntu 16.04 platform, only
- Mitaka
- Liberty
- Kilo, on Contrail networking only

Installation Prerequisites

Before installing Server Manager ensure the following prerequisites are met.

- The system has Internet access to get dependent packages. Ensure access is available to the Ubuntu archive **mirrors/repos** at **/etc/apt/sources.list**.



NOTE: Server Manager is tested with only the following versions of dependent packages: Ansible 2.2.0.0, Docker 1.13.0, Puppet 3.7.3-1, and Cobbler 2.6.3-1. These tested versions are installed during Server Manager installation.

- Puppet Master requires the fully-qualified domain name (FQDN) of the Server Manager for key generation. The domain name is taken from the **/etc/hosts** file. If the server is part of multiple domains, specify the domain name by using the **--domain** option during the installation.
- On multi-interface systems, specify the interface on which Server Manager needs to listen by using the **--hostip** option during installation. If the listening interface is not specified, the first available interface from the **ifconfig** list is used.
- The system administrator might need to configure the Linux kernel security module AppArmor to allow **server-manager** access.

Installing Server Manager

Server Manager and all of its components (Server Manager, monitoring, Server Manager client, Server Manager Web user interface) are provided together in a wrapper installation package:

Ubuntu: **contrail-server-manager-installer_<version>-<sku>.deb**

You can choose to install all components at once or install individual components one at a time.

Use the following steps to install and set up Server Manager and its components.

1. Install the Server Manager packages:

Ubuntu: **dpkg --i contrail-server-manager-installer_<version>-<sku>.deb**



NOTE: Make sure to select the correct version package that corresponds to the platform for which you are installing.

2. Set up the Server Manager components. Use the **setup.sh** command to install all of the components, or you can install individual components.

```
cd /opt/contrail/contrail_server_manager; ./setup.sh [--hostip=<ip address>]
[--domain=<domain name>]
```

- To set up all components:

```
./setup.sh --all
```

- To set up only the Server Manager server:

```
./setup.sh --sm=contrail-server-manager_<version-sku>.deb
```

- To set up only the Server Manager client:

```
setup.sh --sm-client=contrail-server-manager_<version-sku>.deb
```

- To set up only the Server Manager user interface:

```
setup.sh --webui=contrail-server-manager_<version-sku>.deb
```

- To set up only Server Manager monitoring:

```
setup.sh --sm-mon=contrail-server-manager_<version-sku>.deb
```

Other options include:

- --sm-cliff-client
- --nowebui
- --nosm-mon

3. Installation logs are located at **/var/log/contrail/install_logs/**.

Finishing the Installation

The Server Manager service does not start automatically upon successful installation. You must finish the installation by modifying the following templates. Refer to the sample configuration section included in this topic for details about configuring these files.

```
/etc/cobbler/dhcp.template
/etc/cobbler/named.template
/etc/bind/named.conf.options
/etc/cobbler/settings
/etc/cobbler/modules.conf
/etc/mail/sendmail.cf
```

Starting the Server Manager Service

When you are finished modifying the templates to match your environment, start the Server Manager service using the following command:

```
service contrail-server-manager start
```

Upgrading Server Manager Software

If you are upgrading Server Manager software from a previous version to the current version, use the following guidelines to ensure successful installation.

Steps for Upgrading

Use the following steps to upgrade your Server Manager installation.



NOTE: You do not need to manually delete your previous Server Manager installation before upgrading.

1. `dpkg -i <contrail-server-manager-installer*.deb>`
2. `cd /opt/contrail/contrail_server_manager`
3. `./setup.sh -all`
4. After the setup script has completed running, you can restart Server Manager by issuing:

```
service contrail-server-manager restart
```

It is not necessary to reconfigure the templates of DHCP, bind, and so on. Previous template configurations and configured data are preserved during the upgrade.

Server Manager Installation Completion Checks

The following are various checks you can use to investigate the status of your Server Manager installation.

Server Manager Checks

Use the following to check that the Server Manager installation is complete.

- Use the following commands to verify that the services are running:

```
service contrail-server-manager status
```

```
service cobblerd status
```

```
cobbler sync
```

```
service bind9 status
```



```
service isc-dhcp-server status
```

```
service apache2 status
```

```
service docker status
```

- Also verify processes using the following command:

```
ps auwx | grep Passenger
```

Server Manager Client Checks

- Verify the items listed:

```
which server-manager
```

- Check the client configuration at `/etc/contrail/sm-client-config.ini`
- Make sure that `listen_ip_addr` is configured with the correct Server Manager IP address.

Server Manager WebUI Checks

- Verify the status of the Server Manager WebUI:

```
service supervisor-webui-sm status
```

- Check the webui access from the browser:
 - Contrail release 4.0 and greater—`http:<server manager ip> :9143`
 - Contrail releases 3.0, 3.1, and 3.2—`http:<server manager ip> :9080`
 - Contrail release 2.2 and lower—`http:<server manager ip> :8080`

Sample Configurations for Server Manager Templates

The following are sample parameters for the Server Manager templates. Use settings specific for your environment. Typically, you configure parameters for DHCP, bind, and e-mail services.

Sample Settings

```
bind_master: 10.XX.11.6
```

```
manage_forward_zones: ['contrail.juniper.net']
```

```
manage_reverse_zones: ['10.XX.11']
```

```
next_server: 10.XX.11.6
```

```
server: 10.XX.11.6
```

Sample dhcp.template File

Add Server Manager hooks into the `dhcp.template` file, so that when DHCP actions occur, such as commit, release, or expire, the Server Manager is notified. The DHCP servers are detected on the Server Manager and the *Discovered* status is maintained.

Use the following sample to help define the subnet blocks that the DHCP server needs to support:

<https://github.com/Juniper/contrail-server-manager/blob/master/src/cobbler/dhcp.template>



NOTE: Your DHCP template must have a separate block for each subnet for which Server Manager will be the DHCP server.

Sample named.conf.options File

Use the following sample to help configure the `/etc/bind/named.conf.options`:

<https://github.com/Juniper/contrail-server-manager/blob/master/src/cobbler/named.conf.options.u>

You can also configure the following parameter:

```
forwarders {  
    0.0.0.0;  
};
```

Sample named.template File

Use the following sample to help configure the `/etc/cobbler/named.template`:

<https://github.com/Juniper/contrail-server-manager/blob/master/src/cobbler/named.template>

The sendmail.cf File

The `sendmail.cf` template is present with a `juniper.net` configuration. Populate it with configuration specific to your environment. The Server Manager uses the template to generate e-mails when reimaging or provisioning is completed.

Related Documentation

- [Using Server Manager to Automate Provisioning on page 178](#)
- [Using the Server Manager Web User Interface on page 206](#)
- [Installing and Using Server Manager Lite on page 225](#)

Using Server Manager to Automate Provisioning

- [Overview of Server Manager on page 179](#)
- [Server Manager Requirements and Assumptions on page 179](#)
- [Server Manager Component Interactions on page 181](#)
- [Configuring Server Manager on page 182](#)
- [Configuring the Cobbler DHCP Template on page 184](#)
- [User-Defined Tags for Server Manager on page 184](#)
- [Server Manager Client Configuration File on page 185](#)

- [Restart Services on page 185](#)
- [Accessing Server Manager on page 186](#)
- [Communicating with the Server Manager Client on page 187](#)
- [Server Manager Commands for Configuring Servers on page 187](#)
- [Server Manager REST API Calls on page 200](#)
- [Example: Reimaging and Provisioning a Server on page 205](#)

Overview of Server Manager

The Contrail Server Manager is used to provision, configure, and reimage a Contrail virtual network system of servers, clusters, and nodes.

This section describes the functions and usage guidelines for the Contrail Server Manager.

The Server Manager provides a simple, centralized way for users to manage and configure components of a virtual network system running across multiple physical and virtual servers in a cloud infrastructure.

You can use Server Manager to configure, provision, and reimage servers with the correct software version and packages for the nodes that are running on each server in multiple virtual network system clusters.

The Server Manager:

- Provides REST APIs to handle customer requests.
- Manages its own database to store information about the servers.
- Interacts with other open source products such as Cobbler, Puppet, and Ansible to configure servers based on user requests.

Server Manager Requirements and Assumptions

The following are requirements and assumptions for the Server Manager:

- The Server Manager runs on a Linux server (bare metal or virtual machine) and assumes availability of several software products with which it interacts to provide the functionality of managing servers.
- The Server Manager has network connectivity to the servers it is trying to manage.
- The Server Manager has access to a remote power management tool to power cycle the servers that it manages.
- The Server Manager uses Cobbler software for Linux provisioning to configure and download software to physical servers. Cobbler resides on the same server that is running the Server Manager daemon.

- Server Manager assumes that DNS and DHCP servers embedded with Cobbler provide IP addresses and names to the servers being managed, although it is possible to use external DNS and DHCP servers.
- The Server Manager uses Puppet software, an open source configuration management tool, to accomplish the configuration management of target servers, including the installation and configuration of different software packages and the launching of various services.
- Starting with Contrail Release 4.0, Server Manager uses Ansible software, an open source configuration management tool primarily used to automate the configuration and provisioning of Contrail components inside containers.
- The Server Manager also uses Docker to load and move these Contrail containers to the target servers. The Server Manager maintains a local registry on the Server Manager machine and users also have an option to use an external registry from which they can copy their Contrail Docker images directly onto the target servers.
- SQLite3 database management software is used to maintain and manage server configurations and it runs on the same machine where the Server Manager daemon is running.
- Because the server-manager process listens on port 9001, and the server-manager webui listens on ports 9080 and 9143, the firewall must be enabled for those ports.
- Server Manager needs a minimum of 4GB of RAM, 2 CPU cores, and 80GB of disks (to support multiple Contrail installations).
- Server Manager assumes that SSH is enabled on target nodes.

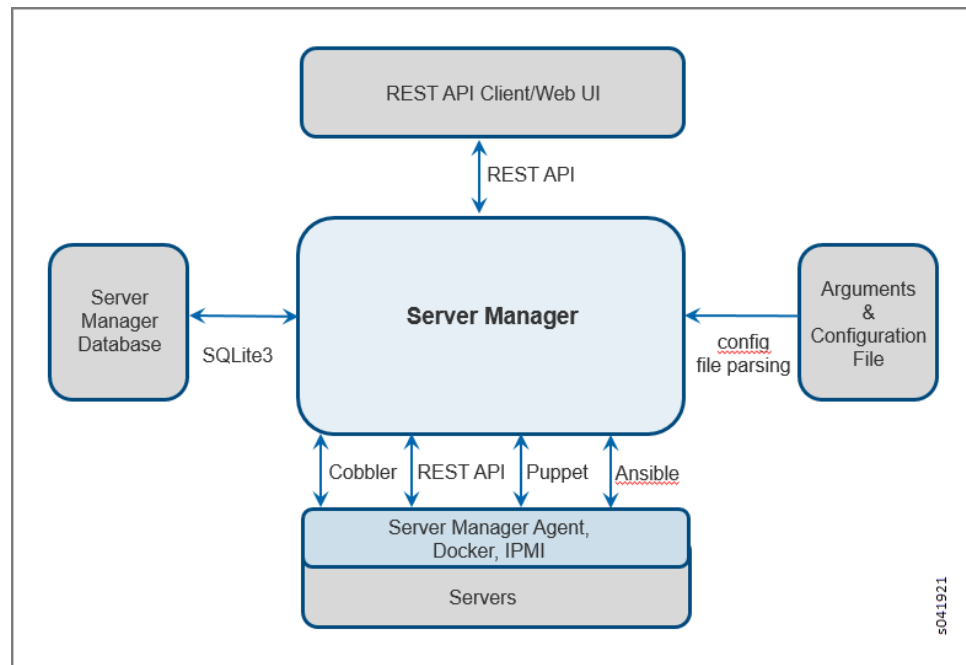
Server Manager Component Interactions

The Server Manager runs as a daemon and provides REST APIs for interaction with the client. The Server Manager accepts user input in the form of REST API requests, performs the requested function on the resources, and responds with a REST API response.

Configuration parameters required by the Server Manager are provided in the Server Manager configuration file. However, the parameters can be overridden by Server Manager command line parameters.

Figure 33 on page 181 illustrates several high-level components with which the Server Manager interacts.

Figure 33: Server Manager Component Interactions



Internally, the Server Manager uses a SQLite3 database to hold server configuration information. The Server Manager coordinates the database configuration information and user requests to manage the servers defined in the database.

While managing the servers, the Server Manager also communicates with other software components. It uses Cobbler for reimaging target servers, Docker to host Contrail containers, and Ansible and Puppet for provisioning, thereby ensuring necessary software packages are installed and configured, required services are running, and so on.

A Server Manager agent runs on each of the servers and communicates with the Server Manager, providing the information needed to monitor the operation of the servers. The Server Manager agent also uses REST APIs to communicate with the Server Manager, and it can use other software tools to fetch other information, such as Intelligent Platform Interface (IPMI). Monitoring functionality is enabled by default with Server Manager installation but can be skipped if the user wishes.

Configuring Server Manager

When the installation of all Server Manager components and dependent packages is finished, configure the Server Manager with parameters that identify your environment and make it available for clients to serve REST API requests.

Upon installation, a sample Server Manager configuration file is created at:

`/opt/contrail/server_manager/sm-config.ini`

Modify the **sm-config.ini** configuration file to include parameter values specific to your environment.

The environment-specific configuration section of the **sm-config.ini** file is named **SERVER-MANAGER**.

The following example shows the format and parameters of the **SERVER-MANAGER** section. Typically, only the **listen_ip_addr**, **cobbler_username**, and **cobbler_passwd** values need to be modified.

```
[SERVER-MANAGER]

listen_ip_addr = <SM-IP-address>

listen_port    = <port-number>

cobbler_ip_address = <cobbler-ip-address>
cobbler_port     = <cobbler-port-number>
cobbler_username = <cobbler-username>
cobbler_password = <cobbler-password>

ipmi_username = <IPMI username>
ipmi_password = <IPMI password>
ipmi_type     = <IPMI type>
```

[Table 8 on page 182](#) provides details for each of the parameters in the **SERVER-MANAGER** section.

Table 8: Server Manager Parameters

Parameter	Configuration
listen_ip_addr	Specify the IP address of the server on which the Server Manager is listening for REST API requests.
listen_port	The port number on which the Server Manager is listening for REST API requests. The default is 9001.

Table 8: Server Manager Parameters (*continued*)

Parameter	Configuration
<code>cobbler_ip_address</code>	The IP address used to access Cobbler. This address MUST be the same address as the <code>listen_ip_address</code> . The Server Manager assumes that the Cobbler service is running on the same server as the Server Manager service.
<code>cobbler_port</code>	The port on which Cobbler listens for user requests. Leave this field blank.
<code>cobbler_username</code>	Specify the user name to access the Cobbler service. Specify testing unless your Cobbler settings have been modified to use a different user name.
<code>cobbler_password</code>	Specify the password to access the Cobbler service. Specify testing unless your Cobbler settings have been modified to use a different password.
<code>ipmi_username</code>	The IPMI username for power management.
<code>ipmi_password</code>	The IPMI password for power management.
<code>ipmi_type</code>	The IPMI type (ipmilan, lanplus, or other Cobbler-supported types).

Starting with Contrail Release 4.0, there is an **ANSIBLE-SERVER** section for parameters for running the Server Manager Ansible daemon, which is used to set up a Docker registry. This registry is used by Ansible to provision Contrail Release 4.0 containers onto targets. These values can be modified to reflect any remote or non-Server Manager Docker registry that the user wants to use to host the Contrail Release 4.0 Docker containers. The following example shows the format and parameters of the **ANSIBLE-SERVER** section:

```
[ANSIBLE-SERVER]

docker_insecure_registries = <IP address:Port>

docker_registry = <IP address:Port>

ansible_srvr_ip = <IP address>

ansible_srvr_port = <Port>

ansible_log_path = /var/log/contrail-server-manager/debug.log
```

[Table 9 on page 183](#) provides details for each of the parameters in the **ANSIBLE-SERVER** section.

Table 9: Ansible Server Parameters

Parameter	Configuration
<code>docker_insecure_registries</code>	Specify the IP address and port of the server on which the insecure Docker registry used by the Server Manager resides
<code>docker_registry</code>	Specify the IP address and port of the server on which the Docker registry used by the Server Manager resides

Table 9: Ansible Server Parameters (*continued*)

Parameter	Configuration
<code>ansible_svr_ip</code>	Specify the IP address of the Server Manager machine on which the Ansible daemon will run
<code>ansible_svr_port</code>	Specify the port on the Server Manager machine on which the Ansible daemon will run
<code>ansible_log_path</code>	Specify the log path where the Ansible daemon stores its log messages

Configuring the Cobbler DHCP Template

In addition to configuring the `sm_config.ini` file, you must manually change the settings in the `/etc/cobbler/dhcp.template` file to use the correct subnet address, mask, and DNS domain name for your environment. Optionally, you can also restrict the use of the current instance of Server Manager and Cobbler to a subset of servers in the network.

The following is a link to a sample `dhcp.template` file, which you can modify to match the subnets in your setup.



NOTE: The IP addresses and other values in the sample are for example purposes only. Be sure to use values that are correct for your environment.

Sample dhcp.template

<https://github.com/Juniper/contrail-server-manager/blob/master/src/cobbler/dhcp.template.u.sample>

User-Defined Tags for Server Manager

Server Manager enables you to define tags that can be used to group servers for performing a particular operation, such as show information, reimage, provision, and so on. Server Manager supports up to seven different tags that can be configured and used for grouping servers.

The names of user-defined tags are kept in the `tags.ini` file, at `/etc/contrail_smgr/tags.ini`.

It is possible to modify tag names, and add or remove tags dynamically using the Server Manager REST API interface. However, if a tag is already being used to group servers, the tag must be removed from the servers before tag modification is allowed.

The following is a sample `tags.ini` file that is copied on installation. In the sample file, five tags are defined – `datacenter`, `floor`, `hall`, `rack`, and `user_tag`. Use the tags to group servers together.

```
[TAGS]
tag1 = datacenter
tag2 = floor
tag3 = hall
tag4 = rack
tag5 = user_tag
```


Server Manager Client Configuration File

The Server Manager client application installation copies the `/etc/contrail/sm-client-config.ini` sample configuration file. The sample file contains parameter values such as the IP address to reach the Server Manager and the port used by Server Manager. You must modify the values in the `sm-client-config.ini` file to match your environment.

The **CLUSTER** and **SERVER** subsections have fields that represent the password for a host or a service. If a value for the password field is not explicitly provided, the Server Manager selects a default password.

Starting with Contrail Release 3.0.2, if you don't explicitly specify a password, a password is automatically generated by the system. This makes the clusters provisioned by Server Manager more secure. There are no default passwords. The system administrator can specify the passwords to configure, or you can use the passwords that are automatically generated by Server Manager.

The following fields get an autogenerated password whenever an explicit password is not provided.

- Ceilometer Mongoddb password
- Ceilometer keystone auth password
- Cinder keystone auth password
- Glance keystone auth password
- Heat encryption key
- Heat keystone auth password
- Keystone admin password
- Keystone admin token
- MYSQL root password
- MYSQL service password
- Neutron keystone auth password
- Nova keystone auth password
- Swift keystone auth password

Restart Services

When all user changes have been made to the configuration files, restart the Server Manager so that it runs with the modifications:

service contrail-server-manager restart

Accessing Server Manager

When the Server Manager configuration has been customized to your environment, and the required daemon services are running, clients can request and use services of the Server Manager by using REST APIs. Any standard REST API client can be used to construct and send REST API requests and process Server Manager responses.

The following steps are typically required to fully implement a new cluster of servers being managed by the Server Manager.

1. Add a boot image (ISO) to server-manager, along with the kickstart and preseed files compatible with your datacenter server. Each Server Manager release has a default kickstart file. If your system administrator doesn't provide the kickstart files, Server Manager default files will be used.
2. Add the Contrail image you are using to Server Manager.
3. Add the cluster(s) to Server Manager. You can add common provisioning parameters for servers to the cluster, and the parameters get passed to the server when provisioning starts.

4. Add the servers that will be managed by Server Manager. Remember to add the **cluster_id** to link with the cluster.

The following are the minimum parameters needed for reimaging or provisioning:

- ID
 - cluster
 - domain
 - interface details
 - roles assigned to each server
 - password
5. Specify the name and location of boot images, packages, and repositories used to bring up the servers with needed software of the supported versions.
 6. Provision or configure the servers by installing necessary packages, creating configuration files, and bringing up the correct services so that each server can perform the functions or role(s) configured for that server.

A Contrail system of servers has several components or roles that work together to provide the functionality of the virtual network system, including: control, config, analytics, compute, web-ui, OpenStack, and database. Each of the roles has different requirements for the software and services needed. The provisioning REST API enables the client to configure the roles on servers using the Server Manager.

7. Set up API calls for monitoring servers.

Once the servers in the Contrail system are correctly reimaged and provisioned to run configured roles, the server monitoring REST API calls allow clients to monitor performance of the servers as they provide one or more role functions.

Communicating with the Server Manager Client

Server Manager provides a REST API interface for clients to talk to the Server Manager software. Any client that can send and receive REST API requests and responses can be used to communicate with Server Manager, for example, Curl or Postman. Additionally, the Server Manager software provides a client with a simplified CLI interface, in a separate package. The Server Manager client can be installed and run on the Server Manager machine itself or on another server with an IP connection to the Server Manager machine.

Prior to using the Server Manager client CLI commands, you need to modify the **sm-client-config.ini** file to specify the IP address and the port for the Server Manager.

Each of the commands described in this section takes a set of parameters you specify, constructs a REST API request to the Server Manager, and provides the server's response.

The following describes each Server Manager client CLI command in detail.

Server Manager Commands for Configuring Servers

This section describes commands that are used to configure servers and server parameters in the Server Manager database. These commands allow you to add, modify, delete, or view servers, clusters, images, and tags.

- [Server Manager Commands Common Options on page 187](#)
- [Add New Servers or Update Existing Servers on page 188](#)
- [Delete Servers on page 189](#)
- [Display Server Configuration on page 189](#)
- [Server Manager Commands for Managing Clusters on page 190](#)
- [Server Manager Commands for Managing Tags on page 191](#)
- [Server Manager Commands for Managing Images on page 193](#)
- [Server Manager Operational Commands for Managing Servers on page 196](#)
- [Reimaging Server\(s\) on page 196](#)
- [Provisioning and Configuring Roles on Servers on page 197](#)
- [Restarting Server\(s\) on page 198](#)
- [Show Status of Server\(s\) on page 199](#)
- [Show Status of Provision on page 199](#)

Server Manager Commands Common Options

The common options in [Table 10 on page 188](#) are available with every Server Manager command.

Table 10: Common Command Options

Option	Description
-h, --help	Show the options available for the current command and exit.
--config_file CONFIG_FILE, -c CONFIG_FILE	The name of the Server Manager client configuration file. The default file is <code>/etc/contrail/sm-client-config.ini</code> .
--smgr_ip SMGR_IP	The IP address of the Server Manager process if different from that specified in the config file.
--smgr_port SMGR_PORT	The port that the Server Manager process is listening on if different from that in the config file.

Add New Servers or Update Existing Servers

Use the **server-manager add** command to create a new server or update a server in the Server Manager database.

```
server-manager [-h] [--smgr_ip SMGR_IP] [--smgr_port SMGR_PORT]
               [--config_file CONFIG_FILE] add server [-f FILE_NAME]
```

Table 11 on page 188 lists the optional arguments.

Table 11: Server Manager Add Server Command Options

Option	Description
--file_name FILE_NAME, -f FILE_NAME	The JSON file that contains the server parameter values.

The JSON file contains a number of server entries, in the format shown in the following example:

<https://github.com/Juniper/contrail-server-manager/blob/R3.1/src/client/new-server.json>

Most of the parameters in the JSON sample file are self-explanatory. **Cluster_id** defines the cluster to which the server belongs. The sample **roles** array in the example lists all valid role values. **Tag** defines the mapping of tag names and values for grouping and classifying the server.

The **server-manager add** command will add a new entry if the server with the given ID or `mac_address` does not exist in the Server Manager database. If an entry already exists, the add command modifies the fields in the existing entry with any new parameters specified.



NOTE: It is not possible to re-add an existing MAC address under a new server, even if the ID and IP address of that new server are unique.

Delete Servers

Use the **server-manager delete** command to delete one or more servers from the Server Manager database.

```
server-manager [-h] [--smgr_ip SMGR_IP] [--smgr_port SMGR_PORT][--config_file
CONFIG_FILE] delete server (--server_id SERVER_ID | --mac MAC | --ip IP |
--cluster_id CLUSTER_ID | --tag <tag_name=tag_value>.. )
```

Table 12 on page 189 lists the optional arguments.

Table 12: Server Manager Delete Server Command Options

Option	Description
--server_id SERVER_ID	The server ID for the server or servers to be deleted.
--mac MAC	The MAC address for the server or servers to be deleted.
--ip IP	The IP address for the server or servers to be deleted.
--cluster_id CLUSTER_ID	The cluster ID for the server or servers to be deleted.
--tag TagName=TagValue	The TagName that is to be matched with the Tagvalue. Up to seven TagName and Tagvalue pairs separated by commas can be provided.

The criteria for identifying servers to be deleted can be specified by providing the **server_id** or the server: **mac address**, **ip**, **cluster_id**, or the **TagName = TagValue**.

Provide one of the server matching criteria to display a list of servers available to be deleted.

Display Server Configuration

Use the **server-manager display** command to display the configuration of servers from the Server Manager database.

```
server-manager display [--smgr_ip SMGR_IP] [--smgr_port SMGR_PORT][--config_file
CONFIG_FILE]
server (--server_id SERVER_ID | --mac MAC | --ip IP |
--cluster_id CLUSTER_ID | --tag <tag_name=tag_value>.. ) [--detail]
```

Table 13 on page 189 lists the optional arguments.

Table 13: Server Manager Display Server Command Options

Option	Description
--server_id SERVER_ID	The server ID for the server or servers to be deleted.
--mac MAC	The MAC address for the server or servers to be displayed.
--ip IP	The IP address for the server or servers to be displayed.

Table 13: Server Manager Display Server Command Options (*continued*)

Option	Description
<code>--cluster_id CLUSTER_ID</code>	The cluster ID for the server or servers to be displayed.
<code>--tag TagName=TagValue</code>	The TagName that is to be matched with the Tagvalue. Up to seven TagName and Tagvalue pairs separated by commas can be provided.
<code>--detail, -d</code>	Flag to indicate if details are requested.

The criteria for identifying servers to be displayed can be specified by providing the **server_id** or one of the following server parameters: **mac address**, **ip**, **cluster_id**, or **TagName=TagValue**.

Provide one or more of the server matching criteria to display a list of servers.

Server Manager Commands for Managing Clusters

A cluster is used to store parameter values that are common to all servers belonging to that cluster. The commands in this section facilitate managing clusters in the Server Manager database, enabling you to add, modify, delete, and view clusters.



NOTE: Whenever a server is created with a specific **cluster_id**, Server Manager checks to see if a cluster with that ID has already been created. If there is no matching **cluster_id** already in the database, an error is returned.

- [Create a New Cluster or Update an Existing Cluster on page 190](#)
- [Delete a Cluster on page 191](#)
- [Display Cluster Configuration on page 191](#)

Create a New Cluster or Update an Existing Cluster

Use the **server-manager add cluster** command to create a new cluster or update an existing cluster in the Server Manager database.

```
server-manager add cluster [--file_name FILE_NAME]
```

[Table 14 on page 190](#) lists the optional arguments.

Table 14: Server Manager Add Cluster Command Options

Option	Description
<code>--file_name FILE_NAME, -f FILE_NAME</code>	The JSON file that contains the cluster parameter values.

The JSON file contains a number of cluster entries, in the format shown in the following example:

<https://github.com/Juniper/contrail-server-manager/blob/master/src/client/new-cluster-contrail-4.x.json>

Server membership to a cluster is determined by specifying the ID corresponding to the cluster when defining the server. All of the cluster parameters are available to the server when provisioning roles on the server.

Delete a Cluster

Use the **server-manager delete** command to delete a cluster from the Server Manager database that are no longer needed. Use this command after all servers in the cluster have been deleted.



NOTE: A cluster can only be deleted if no servers are attached to it. If any servers are attached, deletion will fail.

```
server-manager delete cluster [--cluster_id CLUSTER_ID]
```

Table 15 on page 191 lists the optional arguments.

Table 15: Server Manager Delete Cluster Command Options

Option	Description
<code>--cluster_id CLUSTER_ID</code>	The cluster ID for the server or servers to be displayed.

Display Cluster Configuration

Use the **server-manager display** command to list the configuration of a cluster.

```
server-manager display cluster [--cluster_id CLUSTER_ID] [--detail]
```

Table 16 on page 191 lists the optional arguments.

Table 16: Server Manager Display Cluster Command Options

Option	Description
<code>--detail, -d</code>	Flag to indicate if details are requested.
<code>--cluster_id CLUSTER_ID</code>	The cluster ID for the cluster or clusters.

You can optionally specify a cluster ID to get server information about a particular cluster. If the optional parameter is not specified, server information about all clusters in the system is returned.

Server Manager Commands for Managing Tags

Tags are used for grouping servers together so that an operation such as show, reimage, provision, status, and so on can be easily performed on servers that have matching tags. The Server Manager provides a flexible way for you to define your own tags, and then use those tags to assign values to servers. Servers with matching tag values can be easily grouped together. The Server Manager can store a maximum of seven tag values. At initialization, the Server Manager reads the tag names from the configuration file. The tag names can be retrieved or modified using CLI commands. When modifying tag names,

the Server Manager ensures that the tag name being modified is not used by any of the server entries.

- [Create a New Tag or Update an Existing Tag on page 192](#)
- [Display Tag Configuration on page 192](#)

Create a New Tag or Update an Existing Tag

Use the **server-manager add** command to create a new tag or update an existing tag in the Server Manager database.

```
server-manager add tag [--file_name FILE_NAME] [--tags TAG_LIST]
```

[Table 17 on page 192](#) lists the optional arguments.

Table 17: Server Manager Add New Tag

Option	Description
<code>--file_name FILE_NAME, -f FILE_NAME</code>	The JSON file that contains the tag names.
<code>--tags TAG_LIST</code>	Comma separated list of tag number and tag. For example: tag0=abc,tag1=xyz

The sample JSON file contains a number of tag entries, in the format shown in the following example:

```
{
  "tag1" : "data-center",
  "tag2" : "floor",
  "tag3" : "",
  "tag4" : "pod",
  "tag5" : "rack",
}
```

In the example, you specify a JSON file to add or modify the tags, tag1 through tag5. For tag3, the "" value specifies that if the tag is defined prior to the CLI command, it is removed on execution of the command. The tag name for tag1 is set to data-center. This is allowed if, and only if, none of the server entries are using tag1.

Display Tag Configuration

Use the **server-manager display** command to list the configuration of a tag.

```
server-manager display tag
```

The following is sample output for the **display tag** command.

```
{
  "tag1": "datacenter",
  "tag2": "floor",
```



```

    "tag3": "hall",
    "tag4": "rack",
    "tag5": "user_tag"
  }

```

Server Manager Commands for Managing Images

In addition to servers and clusters, the Server Manager also manages information about images and packages that can be used to reimage and configure servers. Images and packages are both stored in the database as images. When new images are added to the database, or existing images are deleted, the Server Manager interfaces with Cobbler to make corresponding modifications in the Cobbler distribution profile for the specified image.

Table 18 on page 193 lists the image types supported.

Table 18: Server Manager Image Types

Image Type	Description
centos	Manages the CentOS ISO base.
contrail-centos-package	Maintains a repository of the package to be installed on the CentOS system image.
ubuntu	Manages the base Ubuntu ISO.
contrail-ubuntu-package	Maintains a repository of packages that contain Contrail and dependent packages to be installed on an Ubuntu base system.
ESXi5.1/ESXi5.5	Manages VMware ESXi 5.1 or 5.5 ISO.

- [Creating New Images or Updating Existing Images on page 193](#)
- [Add an Image on page 194](#)
- [Upload an Image on page 195](#)
- [Delete an Image on page 195](#)
- [Display Image Configuration on page 196](#)

Creating New Images or Updating Existing Images

The Server Manager maintains four types of images – CentOS ISO, Ubuntu ISO, Contrail CentOS package, and Contrail Ubuntu package.

Use the **server-manager add** command or the **server-manager upload** command to add new images to the Server Manager database.

- Use **add** when the new image is present locally on the Server Manager machine. The path provided is the image path on the Server Manager machine.

- Use **upload_image** when the new image is present on the machine where the client program is being invoked. The path provided is the image path on the client machine.

Add an Image

```
server-manager add image [--file_name FILE_NAME]
```

Table 19 on page 194 lists the optional arguments.

Table 19: Server Manager Add Image

Option	Description
<code>--file_name FILE_NAME, -f FILE_NAME</code>	The name of the JSON file that contains the image parameter values.

The JSON file contains an array of possible entries, in the following sample format. The sample shows three images: one CentOS ISO containing Contrail packages, one Ubuntu base ISO, and one Contrail Ubuntu package. When the images are added, corresponding distribution, profile, and repository entries are created in Cobbler by the Server Manager.



NOTE: Release numbers are represented in the sample with `<x.xx>`. Be sure to use the correct release numbers for your image versions.

```
{
  "image": [
    {
      "id": "ubuntu-<x.xx.x>",
      "type": "ubuntu",
      "version": "ubuntu-<x.xx.x>",
      "path": "/iso/ubuntu-<x.xx.x>-server-amd64.iso"
    },
    {
      "id": "centos-<x.xx>",
      "type": "centos",
      "version": "centos-<x.xx>",
      "path": "/iso/CentOS-<x.xx>-x86_64-minimal.iso"
    },
    {
      "id": "contrail-ubuntu-<x.xx>",
```

```

        "type": "contrail-ubuntu-package",
        "version": "contrail-ubuntu-<x.xx>",
        "path": "/iso/contrail-cloud-docker_<x.xx-xx>_all.deb"
    }
]
}

```

Upload an Image

The `server-manager upload_image` command is similar to the `server-manager add` command, except that the path provided for the image being added is the local path on the client machine. This command is useful if the client is being run remotely, not on the Server Manager machine, and the image being added is not physically present on the Server Manager machine.

```
server-manager upload_image image_id image_version image_type file_name
```

[Table 20 on page 195](#) lists the optional arguments.

Table 20: Server Manager Upload Image

Option	Description
<code>image_id</code>	Name of the new image.
<code>image_version</code>	Version number of the new image.
<code>image_type</code>	Type of image: <code>fedora</code> , <code>centos</code> , <code>ubuntu</code> , <code>contrail-ubuntu-package</code> , <code>contrail-centos-package</code>
<code>file_name</code>	Complete path for the file.

Delete an Image

Use the `server-manager delete` command to delete an image from the Server Manager database. When an image is deleted from the Server Manager database, the corresponding distribution, profile, or repository for the image is also deleted from the Cobbler database.

```
server-manager delete image --image_id <image_id>
```

[Table 21 on page 195](#) lists the optional arguments.

Table 21: Server Manager Delete Image

Option	Description
<code>image_id</code>	The image ID for the image to be deleted.

Display Image Configuration

Use the **server-manager display** command to list the configuration of images from the Server Manager database. If the detail flag is specified, detailed information about the image is returned. If the optional **image_id** is not specified, information about all the images is returned.

```
server-manager display image [--image_id IMAGE_ID] [--detail]
```

Table 22 on page 196 lists the optional arguments.

Table 22: Server Manager Display Image Configuration

Option	Description
image_id	The image ID for the image or images.
--detail, -d	Flag to indicate if details are requested.

Server Manager Operational Commands for Managing Servers

The Server Manager commands in the following sections are operational commands for performing a specific operation on a server or a group of servers. These commands assume that the base configuration of entities required to execute the operational commands is already completed using configuration CLI commands.

Reimaging Server(s)

Use the **server-manager reimage** command to reimage a server or servers with a provided base ISO and package. Servers are specified by providing match conditions to select them from the database.

Before issuing the **reimage** command, the images must be added to the Server Manager, which also adds the images to Cobbler. The set of servers to be reimaged can be specified by providing match criteria for servers already added to the Server Manager database, using **server_id**.

You must identify the base image ID to be used to reimage.

The command prompts for a confirmation before making the REST API call to the Server Manager to start reimaging the servers. This confirmation message can be bypassed by specifying the optional **--no_confirm** or **-F** parameter on the command line.

```
server-manager reimage
    [--package_image_id PACKAGE_IMAGE_ID]

    [--no_reboot]

    (--server_id SERVER_ID | --cluster_id CLUSTER_ID | --tag
    <tag_name=tag_value>)

    [--no_confirm]
    base_image_id
```

Options include the following:

Table 23 on page 197 lists the optional arguments.

Table 23: Server Manager Reimage

Option	Description
<code>base_image_id</code>	The image ID of the base image to be used.
<code>--package_image_id PACKAGE_IMAGE_ID, -p PACKAGE_IMAGE_ID</code>	The optional Contrail package to be used to reimage the server or servers.
<code>--no_reboot, -n</code>	Optional parameter to indicate that the server should not be rebooted following the reimage setup.
<code>--server_id SERVER_ID</code>	The server ID for the server or servers to be reimaged.
<code>--cluster_id CLUSTER_ID</code>	The cluster ID for the server or servers to be reimaged.
<code>--tag TagName=TagValue</code>	TagName which is to be matched with Tagvalue
<code>--no_confirm, -F</code>	Flag to bypass confirmation message, default = do NOT bypass.

Provisioning and Configuring Roles on Servers

Use the **server-manager provision** command to provision identified server(s) with configured roles for the virtual network system. The servers can be selected from the database configuration (using standard server match criteria), identified in a JSON file, or provided interactively.

From the configuration of servers in the database, the Server Manager determines which roles to configure on which servers and uses this information along with other parameters from the database to achieve the task of configuring the servers with specific roles.

When the **server-manager provision** command is used, the Server Manager pushes the specified server configurations to the servers.

```
server-manager provision
    (--server_id SERVER_ID | --cluster_id CLUSTER_ID | --tag
    <tag_name=tag_value> )
    [--no_confirm]
    package_image_id
```

Options include the following:

Table 24 on page 197 lists the optional arguments.

Table 24: Server Manager Provision

Option	Description
<code>package_image_id</code>	The Contrail package image ID to be used for provisioning.
<code>--server_id SERVER_ID</code>	The server ID for the server or servers to be provisioned.

Table 24: Server Manager Provision (*continued*)

Option	Description
<code>--cluster_id CLUSTER_ID</code>	The cluster ID for the server or servers to be provisioned.
<code>--tag TagName=TagValue</code>	TagName to be matched with Tagvalue.
<code>--provision_params_file PROVISION_PARAMS_FILE, -f PROVISION_PARAMS_FILE</code>	Optional JSON file containing the parameters for provisioning the server(s).
<code>--no_confirm, -F</code>	Flag to bypass confirmation message, default = do NOT bypass.



NOTE: Adding and deleting roles is not supported in Contrail Release 4.0.

Restarting Server(s)

Use the **server-manager restart** command to reboot identified server(s). Servers can be specified from the database by providing standard match conditions. The **restart** command provides a way to reboot or power-cycle the servers, using the Server Manager REST API interface. If reimaging is intended, use the **restart** command with the **net-boot** flag enabled. When netbooted, the Puppet agent is also installed and configured on the servers. If there are Puppet manifest files created for the server prior to rebooting, the agent pulls those from the Server Manager and executes the configured Puppet manifests. The **restart** command uses an IPMI mechanism to power cycle the servers, if available and configured. Otherwise, the **restart** command uses SSH to the server and the existing reboot command mechanism is used.

```
server-manager restart
  (--server_id SERVER_ID | --cluster_id CLUSTER_ID | --tag
  <tag_name=tag_value>)

  [--net_boot]
  [--no_confirm]
```

Table 25 on page 198 lists the optional arguments.

Table 25: Server Manager Restart

Option	Description
<code>--server_id SERVER_ID</code>	The server ID for the server or servers to be restarted.
<code>--cluster_id CLUSTER_ID</code>	The cluster ID for the server or servers to be restarted.
<code>--tag TagName=TagValue</code>	TagName to be matched with Tagvalue.
<code>--net_boot, -n</code>	Optional parameter to indicate if the server should be netbooted.
<code>--no_confirm, -F</code>	Flag to bypass confirmation message, default = do NOT bypass.

Show Status of Server(s)

Use the **server-manager status** command to view the reimaging or provisioning status of server(s).

```
server-manager status server (--server_id SERVER_ID | --cluster_id CLUSTER_ID
| --tag <tag_name=tag_value>)
```

Table 26 on page 199 lists the optional arguments.

Table 26: Server Manager Status Server

Option	Description
--server_id SERVER_ID	The server ID for the server whose status is to be fetched.
--cluster_id CLUSTER_ID	The cluster ID for the server or servers to be restarted.
--tag TagName=TagValue	TagName to be matched with Tagvalue.

The status command provides a way to fetch the current status of a server.

Status outputs include the following:

```
restart_issued
reimage_started
provision_issued
provision_completed
openstack_started
openstack_completed
```

Show Status of Provision

Use the **server-manager status provision** to view the detailed provisioning status of servers or cluster. The **status** command provides a way to fetch the current status of a provision command.

```
server-manager status provision (--server_id SERVER_ID | --cluster_id
CLUSTER_ID | --tag <tag_name=tag_value>)
```

Table 27 on page 199 lists the optional arguments.

Table 27: Server Manager Status Provision

Option	Description
--server_id SERVER_ID	The server ID for the server whose status is to be fetched.
--cluster_id CLUSTER_ID	The cluster ID for the server or servers to be restarted.
--tag TagName=TagValue	TagName to be matched with Tagvalue.

Server Manager REST API Calls

This section describes all of the REST API calls to the Server Manager. Each description includes an example configuration.

- [REST APIs for Server Manager Configuration Database Entries on page 200](#)
- [API: Add a Server on page 200](#)
- [API: Delete Servers on page 201](#)
- [API: Retrieve Server Configuration on page 201](#)
- [API: Add an Image on page 201](#)
- [API: Upload an Image on page 202](#)
- [API: Get Image Information on page 202](#)
- [API: Delete an Image on page 202](#)
- [API: Add or Modify a Cluster on page 203](#)
- [API: Delete a Cluster on page 203](#)
- [API: Get Cluster Configuration on page 203](#)
- [API: Get All Server Manager Configurations on page 203](#)
- [API: Reimage Servers on page 204](#)
- [API: Provision Servers on page 204](#)
- [API: Restart Servers on page 204](#)

REST APIs for Server Manager Configuration Database Entries

The REST API calls in this section help in configuring different elements in the Server Manager database.



NOTE: The IP addresses and other values in the following are shown for example purposes only. Be sure to use values that are correct for your environment.

API: Add a Server

To add a new server to the service manager configuration database:

URL: `http://<SM-IP-Address>:<SM-Port>/server`

Method: **PUT**

Payload: JSON payload containing an array of servers to be added. For each server in the array, all the parameters are specified as JSON fields. The mask, gateway, password, and domain fields are optional, and if not specified, the values of these fields are taken from the cluster to which the server belongs.

The following is a sample JSON file for adding a server in Contrail Release 4.0.

<https://github.com/Juniper/contrail-server-manager/blob/master/src/client/new-server-contrail-4.x.json>

API: Delete Servers

Use one of the following formats to delete a server.

URL: `http://<SM-IP-Address>:<SM-Port>/server?server_id=SERVER_ID`

`http://<SM-IP-Address>:<SM-Port>/server?cluster_id=CLUSTER_ID`

`http://<SM-IP-Address>:<SM-Port>/server?mac=MAC`

`http://<SM-IP-Address>:<SM-Port>/server?ip=IP`

`http://<SM-IP-Address>:<SM-Port>/server[?tag=<tag_name>=<tag_value>,.]`

Method : DELETE

Payload : None

API: Retrieve Server Configuration

Use one of the following methods to retrieve a server configuration. The detail argument is optional, and specified as part of the URL if details of the server entry are requested.

URL: `http://<SM-IP-Address>:<SM-Port>/server[?server_id=SERVER_ID&detail]`

`http://<SM-IP-Address>:<SM-Port>/server[?cluster_id=CLUSTER_ID&detail]`

`http://<SM-IP-Address>:<SM-Port>/server[?tag=<tag_name>=<tag_value>,.]`

`http://<SM-IP-Address>:<SM-Port>/server[?mac=MAC&detail]`

`http://<SM-IP-Address>:<SM-Port>/server[?ip=IP&detail]`

`http://<SM-IP-Address>:<SM-Port>/server[?tag=<tag_name>=<tag_value>,.]`

Method : GET

Payload : None

API: Add an Image

Use the following to add a new image to the Server Manager configuration database from the Server Manager machine.

An image is either an ISO for a CentOS or Ubuntu distribution or an Ubuntu Contrail package repository. When adding an image, the image file is assumed to be available on the Server Manager machine.

URL : `http://<SM-IP-Address>:<SM-Port>/image`

Method: PUT

Payload: Specifies all the parameters that define the image being added.

See sample payload in the following:

<https://github.com/Juniper/contrail-server-manager/blob/master/src/client/new-package.json>

API: Upload an Image

Use the following to upload a new image from a client to the Server Manager configuration database.

An image is an ISO for a CentOS or Ubuntu distribution or an Ubuntu Contrail package repository. Add image assumes the file is available on the Server Manager, whereas upload image transfers the image file from the client machine to the Server Manager machine.

URL : `http://<SM-IP-Address>:<SM-Port>/image/upload`

Method: **PUT**

Payload: Specifies all the parameters that define the image being added.

```
{
  "image": [
    {
      "id": "Image-id",
      "type": "image_type", <ubuntu or centos or esxi5.1 or esxi5.5 or
      contrail-ubuntu-package or contrail-centos-package>
      "version": "image_version",
      "path": "path-to-image-on-client-machine"
    }
  ]
}
```

API: Get Image Information

Use the following to get image information.

URL : `http://<SM-IP-Address>:<SM-Port>/image[?image_id=IMAGE_ID&detail]`

Method: **GET**

Payload: Specifies criteria for the image being sought. If no match criteria is specified, information about all the images is provided. The details field specifies if details of the image entry in the database are requested.

API: Delete an Image

Use the following to delete an image.

URL: `http://<SM-IP-Address>:<SM-Port>/image?image_id=IMAGE_ID`

Method: **DELETE**

Payload: Specifies criteria for the image being deleted.

API: Add or Modify a Cluster

Use the following to add a cluster to the Server Manager configuration database. A cluster maintains parameters for a set of servers that work together in different roles to provide complete functions for a Contrail cluster.

URL: **http://<SM-IP-Address>:<SM-Port>/cluster**

Method: **PUT**

Payload: Contains the definition of the cluster, including all the global parameters needed by all the servers in the cluster. The `subnet_mask`, `gateway`, `password`, and `domain` fields define parameters that apply to all servers in the VNS. These parameter values can be individually overridden for a server by specifying different values in the server entry.

A sample JSON for Contrail Release 4.0 is at the following:

<https://github.com/Juniper/contrail-server-manager/blob/master/src/client/new-cluster-contrail-4.x.json>

API: Delete a Cluster

Use this API to delete a cluster from the Server Manager database.

URL: **http://<SM-IP-Address>:<SM-Port>/cluster?cluster_id=CLUSTER_ID**

Method: **DELETE**

Payload: None

API: Get Cluster Configuration

Use this API to get a cluster configuration.

URL: **http://<SM-IP-Address>:<SM-Port>/cluster[?cluster_id=CLUSTER_ID&detail]**

Method: **GET**

Payload: None

The optional detail argument is specified as part of the URL if details of the VNS entry are requested.

API: Get All Server Manager Configurations

Use this API to get all configurations of Server Manager objects, including servers, clusters, images, and tags.

URL: **http://<SM-IP-Address>:<SM-Port>/all[?detail]**

Method: **GET**

Payload: None

The optional detail argument is specified as part of the URL if details of the Server Manager configuration are requested.

API: Reimage Servers

Use one of the following API formats to reimage one or more servers.

URL: `http://<SM-IP-Address>:<SM-Port>/server/reimage?server_id=SERVER_ID`
`http://<SM-IP-Address>:<SM-Port>/server/reimage?cluster_id=CLUSTER_ID`
`http://<SM-IP-Address>:<SM-Port>/server/reimage?mac=MAC`
`http://<SM-IP-Address>:<SM-Port>/server/reimage?ip=IP`
`http://<SM-IP-Address>:<SM-Port>/server/reimage [?tag=<tag_name>=<tag_value>,,]`

Method: **POST**

Payload: None

API: Provision Servers

Use this API to provision or configure one or more servers for roles configured on them.

URL: `http://<SM-IP-Address>:<SM-Port>/server/provision`

Method: **POST**

Payload: Specifies the criteria to be used to identify servers which are being provisioned. The servers can be identified by `server_id`, `mac`, `cluster_id` or `tags`. See the following example.

```
{
  server_id : <server_id> OR
  mac : <server_mac_address> OR
  cluster_id : <cluster_id> OR
  tag : {"data-center" : "dc1"}
}
```

API: Restart Servers

This REST API is used to power cycle the servers and reboot either with net-booting enabled or disabled.

If the servers are to be reimaged and reprovisioned, the **net-boot** flag should be set.

If servers are only being reprovisioned, the **net-boot** flag is not needed, however, the Puppet agent must be running on the target systems with the correct puppet configuration to communicate to the puppet master running on the Server Manager.

URL: `http://<SM-IP-Address>:<SM-Port>/server/restart?server_id=SERVER_ID`
`http://<SM-IP-Address>:<SM-Port>/server/restart?[netboot&]cluster_id=CLUSTER_ID`
`http://<SM-IP-Address>:<SM-Port>/server/restart? [netboot&]mac=MAC`
`http://<SM-IP-Address>:<SM-Port>/server/restart? [netboot&]ip=IP`
`http://<SM-IP-Address>:<SM-Port>/server/restart ?`
`[netboot&]tag=<tag_name>=<tag_value>`

Method: **POST**

Payload: Specifies the criteria to be used to identify servers which are being restarted. The servers can be identified by their **server_id**, **mac**, **cluster_id**, or **tag**. The **netboot** parameter specifies if the servers being power-cycled are to be booted from Cobbler or locally.

Example: Reimaging and Provisioning a Server

This example shows the steps used in Server Manager software to configure, reimage, and provision a server running all roles of the Contrail system in a single-node configuration.



NOTE: Component names and IP addresses in the following are used for example only. To use this example in your own environment, be sure to use addresses and names specific to your environment.

The Server Manager client configuration file used for the following CLI commands, is `/opt/contrail/server_manager/client/sm-client-config.ini`. It contains the values for the server IP address and port number as follows:

[SERVER-MANAGER]

`listen_ip_addr = 192.168.1.10 (Server Manager IP address)`

`listen_port = 9001`

Overview

The steps to be followed include:

1. Configure cluster.
2. Configure servers.
3. Configure images.
4. Reimage servers (either using servers configured above or using explicitly specified reimage parameters with the request).
5. Provision servers (either using servers configured above or using explicitly specified provision parameters with the request).

Procedure

1. Configure a cluster.
`server-manager add cluster -f cluster.json`
2. Configure the server.

```
server-manager add server -f server.json
```

3. Configure images.

In the example, the image files for **ubuntu-xx.xx.x** and **contrail-ubuntu-164** are located at the corresponding image path specified on the Server Manager.

```
server-manager add -c smgr_client_config.ini image -f image.json
```

4. Reimage servers.

This step can be performed after the configuration from the previous steps is in the Server Manager database.

```
server-manager reimage --server_id demo-server ubuntu-<x.xx.x>
```

5. Provision servers.

```
server-manager provision --server_id demo-server contrail-ubuntu-164
```



NOTE: The samples for all JSONs used in the procedure above are available as links in the documentation for the API calls for those respective commands.

- See Also**
- [Installing Server Manager on page 173](#)
 - [Using the Server Manager Web User Interface on page 206](#)
 - [Installing and Using Server Manager Lite on page 225](#)

Using the Server Manager Web User Interface

When the Server Manager is installed on your Contrail system, you can also install a Server Manager Web user interface that you can use to access the features of Server Manager.

- [Log In to Server Manager on page 207](#)
- [Create a Cluster for Server Manager on page 207](#)
- [Edit a Cluster through Edit JSON on page 216](#)
- [Working with Servers in the Server Manager User Interface on page 216](#)
- [Add a Server on page 217](#)
- [Edit Tags for Servers on page 219](#)
- [Using the Edit Config Option for Multiple Servers on page 219](#)
- [Edit a Server through Server Manager, Edit JSON on page 220](#)
- [Filter Servers by Tag on page 220](#)
- [Viewing Server Details on page 221](#)

- [Configuring Images and Packages on page 223](#)
- [Add New Image or Package on page 223](#)
- [Selecting Server Manager Actions for Clusters on page 224](#)
- [Reimage a Cluster on page 224](#)
- [Provision a Cluster on page 224](#)

Log In to Server Manager

The Server Manager user interface can be accessed using:

http://<server-manager-user-interface-ip>:9080

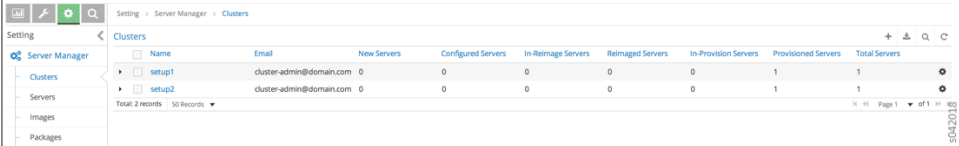
Where **<server-manager-user-interface-ip>** is the IP address of the server on which the Server Manager web user interface is installed.

From the Contrail user interface, select **Setting > Server Manager** to access the Server Manager home page. From this page you can manage Server Manager settings for clusters, servers, images, and packages.

Create a Cluster for Server Manager

Select **Add Cluster** to identify a cluster to be managed by the Server Manager. Select **Setting > Server Manager > Clusters**, to access the **Clusters** page, see [Figure 34 on page 207](#).

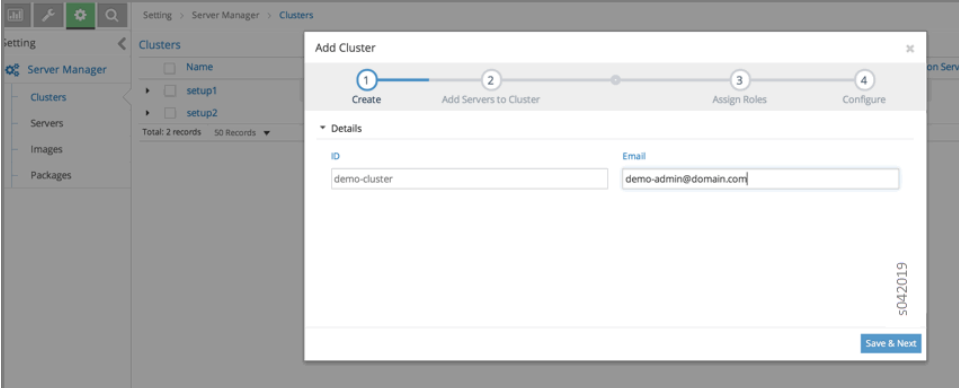
Figure 34: Server Manager > Clusters



Name	Email	New Servers	Configured Servers	In-Reimage Servers	Reimage Servers	In-Provision Servers	Provisioned Servers	Total Servers
setup1	cluster-admin@domain.com	0	0	0	0	0	1	1
setup2	cluster-admin@domain.com	0	0	0	0	0	1	1
Total: 2 records		50 Records						

To create a new cluster, click the plus icon in the upper right of the **Clusters** page. The **Add Cluster** window is displayed. In the **Add Cluster** window, you can add a new cluster ID and the domain e-mail address of the cluster. See [Figure 35 on page 207](#).

Figure 35: Add Cluster



Setting > Server Manager > Clusters

Setting

- Server Manager
- Clusters
- Servers
- Images
- Packages

Clusters

Name	Email	New Servers	Configured Servers	In-Reimage Servers	Reimage Servers	In-Provision Servers	Provisioned Servers	Total Servers
setup1	cluster-admin@domain.com	0	0	0	0	0	1	1
setup2	cluster-admin@domain.com	0	0	0	0	0	1	1
Total: 2 records		50 Records						

Page 1 of 1

Add Cluster

1 Create 2 Add Servers to Cluster 3 Assign Roles 4 Configure

Details

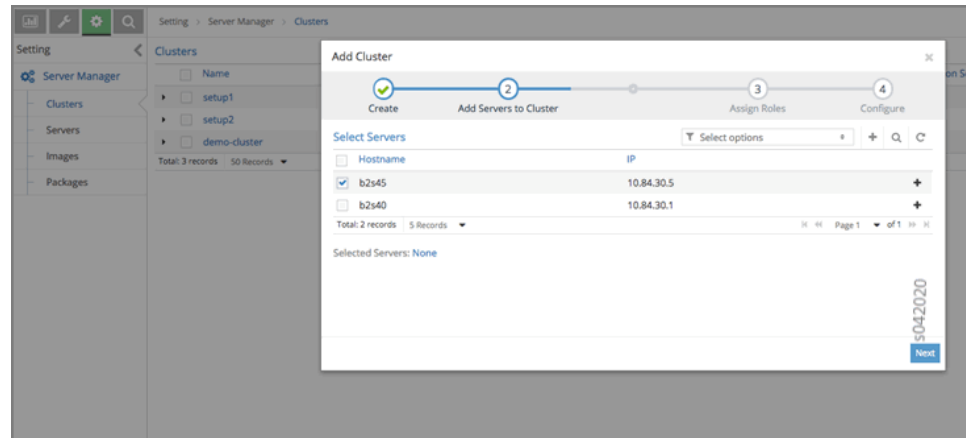
ID: demo-cluster

Email: demo-admin@domain.com

Save & Next

When you are finished adding information about the new cluster in the **Add Clusters** window, click **Save & Next**. Now you can add servers to the cluster, see [Figure 36 on page 208](#).

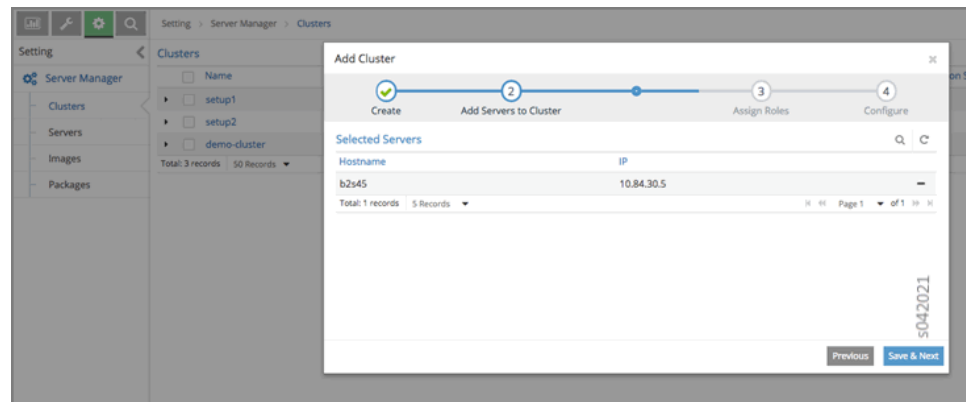
Figure 36: Add Servers to Cluster



Click the check box of each server to be added to the cluster.

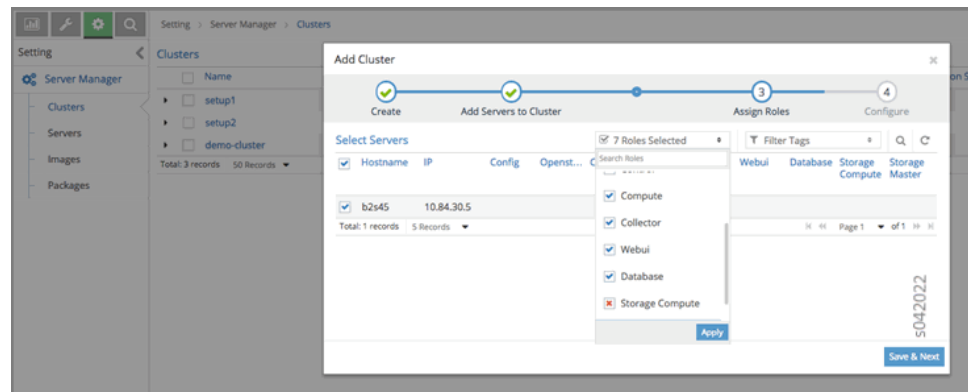
When you are finished, click **Next**. The selected servers are added to the cluster, see [Figure 37 on page 208](#).

Figure 37: Add Servers to Cluster, Next



When you are finished adding servers, click **Save & Next**. Now you can assign Contrail roles to servers that you select in the cluster. Roles available are Config, OpenStack, Control, Compute, and Collector. Select each role assignment for the selected server. You can also unselect any assigned role. The assigned roles correspond to the role functions in operation on the server, see [Figure 38 on page 209](#).

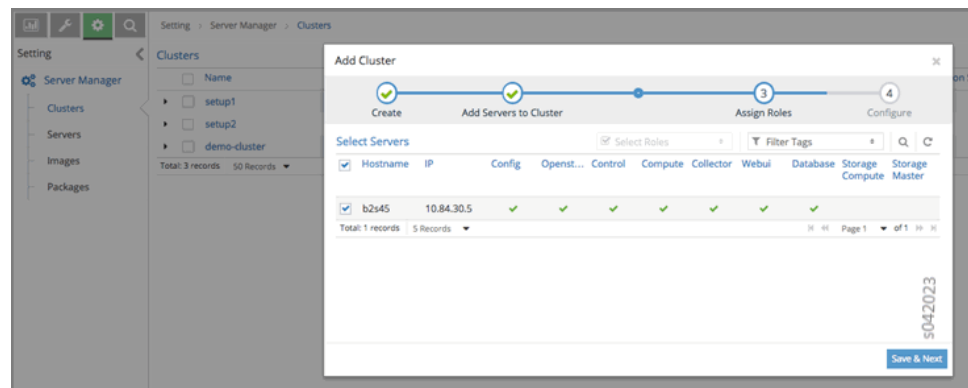
Figure 38: Assign Roles



When you are finished selecting roles for the selected server in the **Roles** window, click **Apply** to save your choices.

Click **Save & Next** to view your selections. Check marks are displayed in the columns of the **Add Cluster** window, see [Figure 39 on page 209](#).

Figure 39: Roles Assigned



The next step after roles are assigned is to enter the cluster configuration information for OpenStack. After viewing the assigned roles, click **Save & Next**. The **Add Cluster** window is displayed. Click an icon that opens a set of fields where you can enter OpenStack or Contrail configuration information for the cluster. In the following image, the **Openstack** icon is selected. You can enter **Keystone** configuration information, such as IP, Admin tenant, user, and password, service tenant, and region name. You can also enable LBaaS and Ceilometer, see [Figure 40 on page 210](#).

Figure 40: OpenStack Configuration

The screenshot shows the Juniper Contrail GUI with the 'Add Cluster' dialog open. The dialog has a progress bar with four steps: Create, Add Servers to Cluster, Assign Roles, and Configure. The 'Configure' step is currently active. Under the 'Openstack' section, the following fields are visible:

- Keystone IP: [Empty]
- Keystone Admin Tenant: [admin]
- Keystone Service Tenant: [services]
- Keystone Admin User: [admin]
- Keystone Region Name: [Empty]
- Keystone Admin Password: [Masked]
- Enable Lbass: [Checked]
- Enable Ceilometer: [Checked]

A 'Save' button is located at the bottom right of the dialog.

In the following image, the Contrail controller icon is selected. You can enter configuration information for Contrail, such as **External BGP, Router ASN, Huge Pages, Core Mask, Encapsulation Priority, Healthcheck Interval, Zookeeper IP Port, Enable SRIOV**, and so on, see [Figure 41 on page 210](#).

Figure 41: Configure Contrail

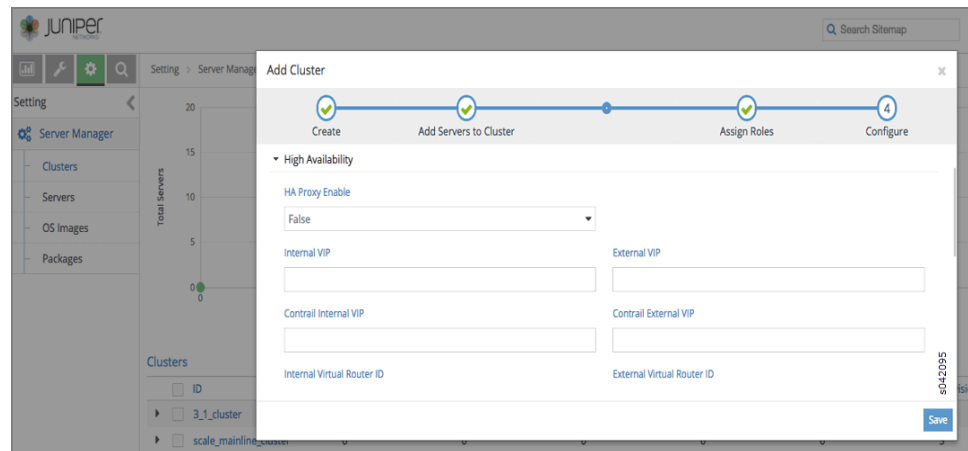
The screenshot shows the Juniper Contrail GUI with the 'Add Cluster' dialog open. The dialog has a progress bar with four steps: Create, Add Servers to Cluster, Assign Roles, and Configure. The 'Configure' step is currently active. Under the 'Contrail Controller' section, the following fields are visible:

- External BGP: [Empty]
- Router ASN: [64512]
- Huge Pages: [Empty]
- Core Mask: [Empty]
- Encapsulation Priority: [VXLAN,MPLSoUDP,MPLSoGRE]
- Healthcheck Interval: [5]
- Zookeeper IP Port: [Empty]
- Enable SRIOV: [Checked]

A 'Save' button is located at the bottom right of the dialog.

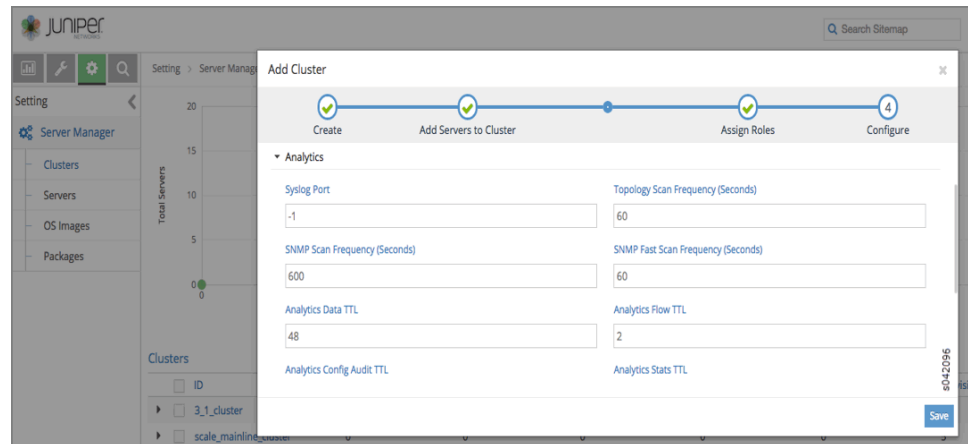
In the following image, the High Availability (HA) icon is selected. You can configure high availability parameters such as **HA Proxy Enable, Internal and External VIP**, and so on, see [Figure 42 on page 211](#).

Figure 42: Configure High Availability



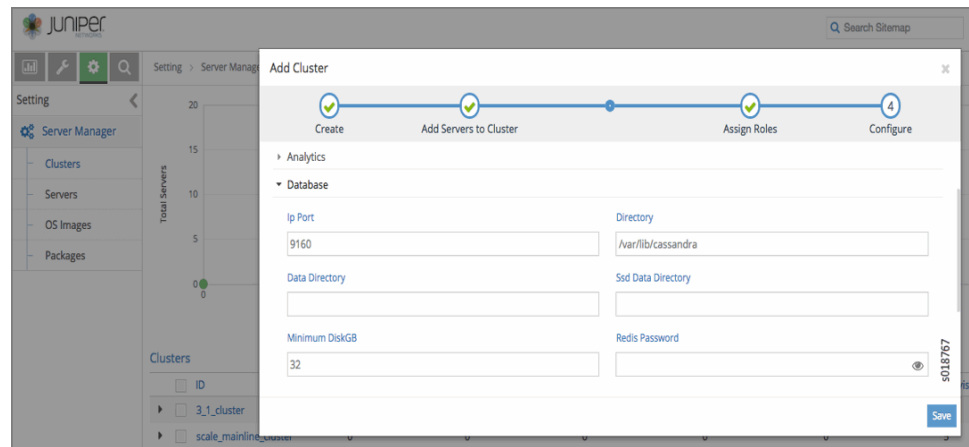
In the following image, the **Analytics** icon is selected. Here you can configure parameters for Contrail Analytics, including **Syslog Port**, various scan frequencies, and various TTL settings, see [Figure 43 on page 211](#).

Figure 43: Configure Analytics



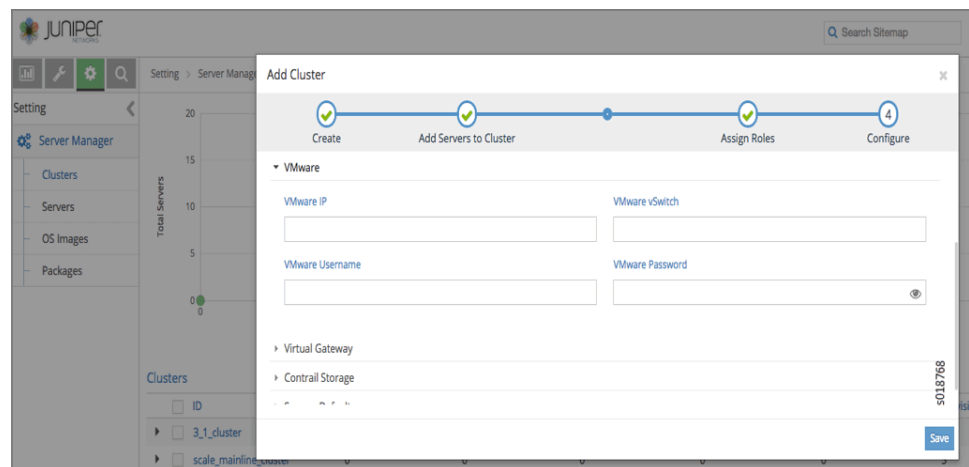
In following image, the **Database** icon is selected. You can configure parameters for the Contrail database, including **IP Port**, **Directory**, **Minimum Disk GB**, and so on, see [Figure 44 on page 212](#).

Figure 44: Configure Database



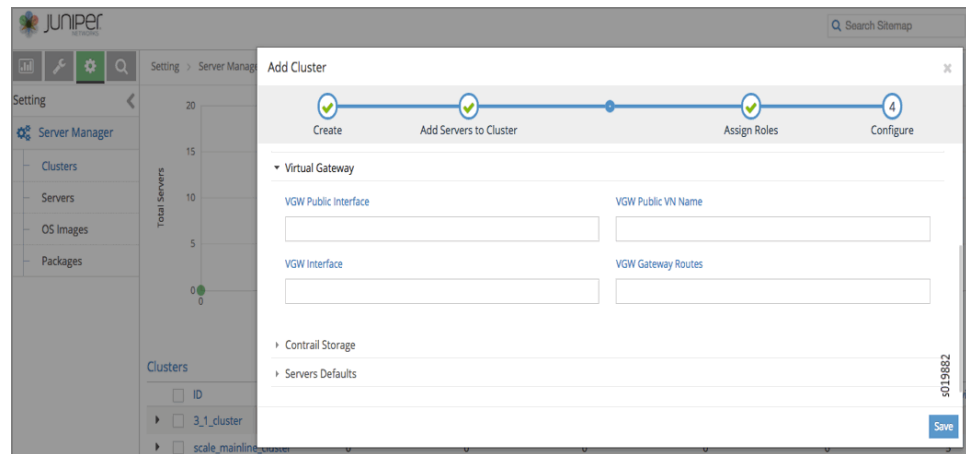
In following image, the **VMware** icon is selected. You can configure parameters for Contrail VMware , including **VMware IP**, **VMware vSwitch**, **Username**, **Password** , and so on, see [Figure 45 on page 212](#).

Figure 45: Configure VMware



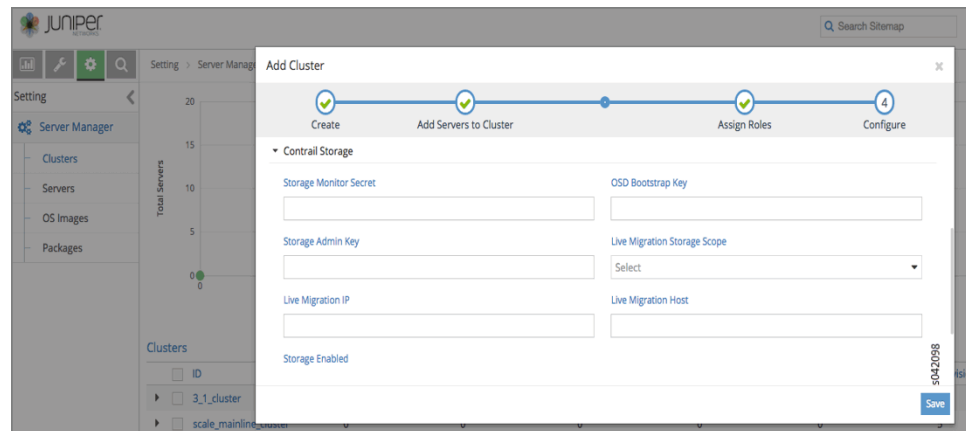
In following image, the **Virtual Gateway** icon is selected. You can configure parameters for the Contrail Virtual Gateway, including **VGW Public Interface**, **VGW Public VN Name**, **VGW Interface**, **Routes** , and so on, see [Figure 46 on page 213](#).

Figure 46: Configure Virtual Gateway



In following image, the **Contrail Storage** icon is selected. You can configure parameters for Contrail Storage, including **Storage Monitor Secret**, **OSD Bootstrap Key**, **Admin Key**, and so on, see [Figure 47 on page 213](#).

Figure 47: Configure Contrail Storage



When you are finished entering all of the cluster configuration information, click **Save** to submit the configurations. You can view all configured clusters on the **Clusters** window by selecting **Setting > Server Manager > Clusters**, see [Figure 48 on page 213](#).

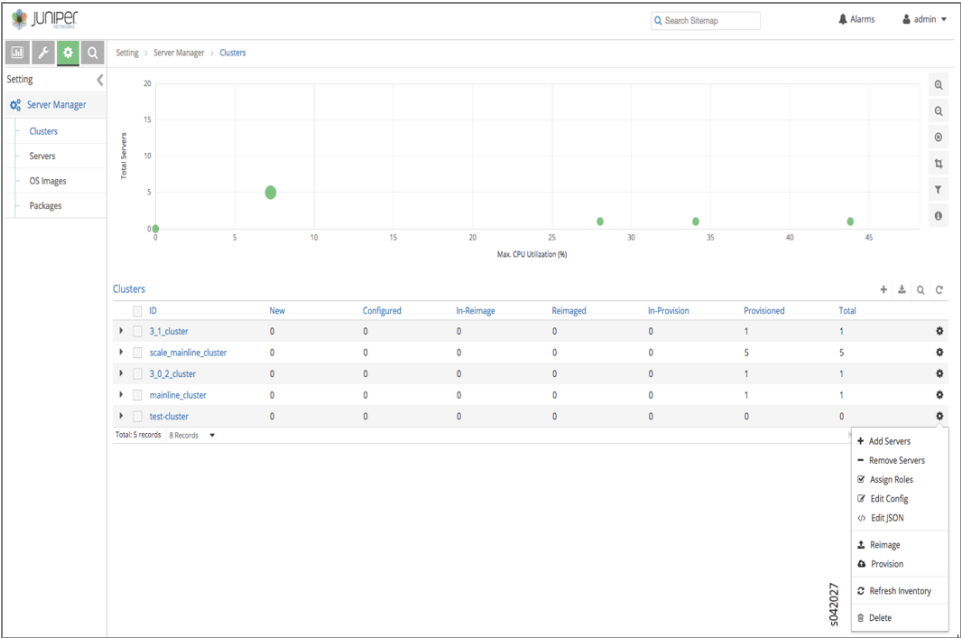
Figure 48: View Configured Clusters

Setting > Server Manager > Clusters								
	Name	Email	New Servers	Configured Servers	In-Reimage Servers	Reimage Servers	In-Provision Servers	Provisioned
	setup1	cluster-admin@domain.com	0	0	0	0	0	0
	setup2	cluster-admin@domain.com	0	0	0	0	0	1
	demo-cluster	demo-admin@domain.com	0	0	0	0	0	1
Total: 3 records			50 Records					

To perform an action on one of the configured clusters, click the gear wheel icon at the right to select from a menu of actions available for that cluster, including **Add Servers**,

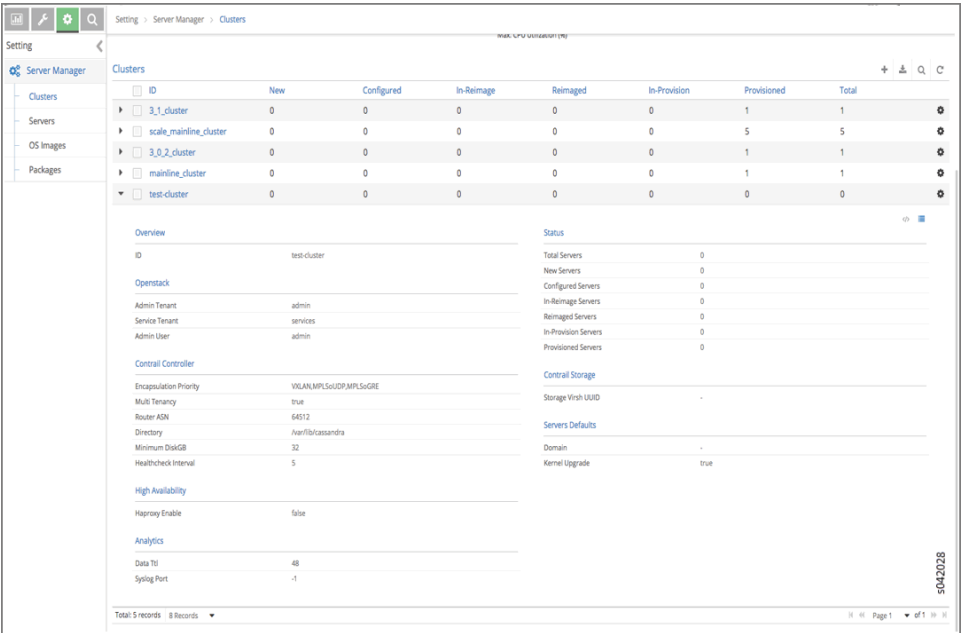
Remove Servers, Assign Roles, Edit Config, Reimage, Provision, and Delete, see [Figure 49 on page 214](#).

Figure 49: Select Cluster Action



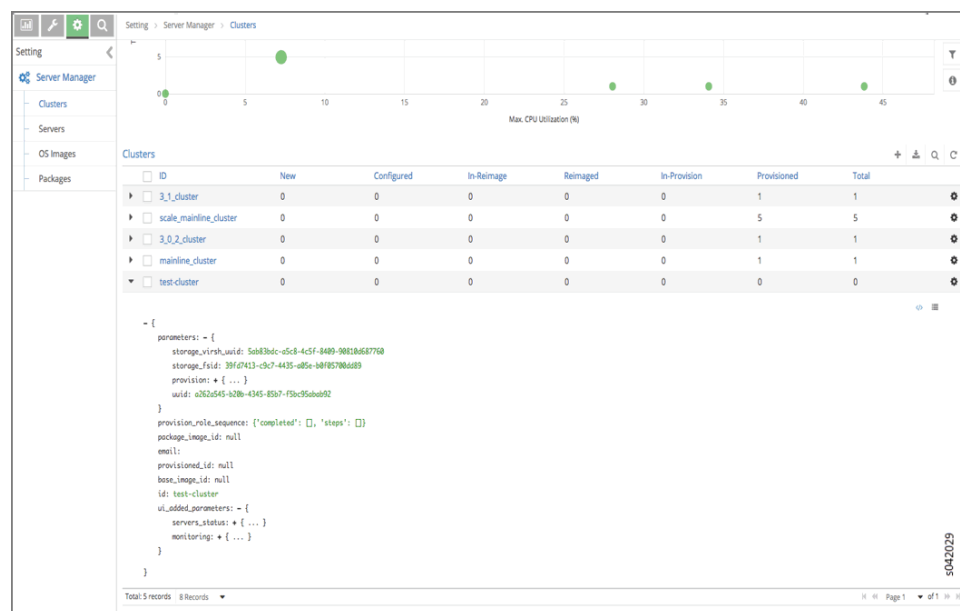
You can also click the expansion icon on the left side of the cluster name to display the details of that cluster in an area below the name line, see [Figure 50 on page 214](#).

Figure 50: Display Cluster Details



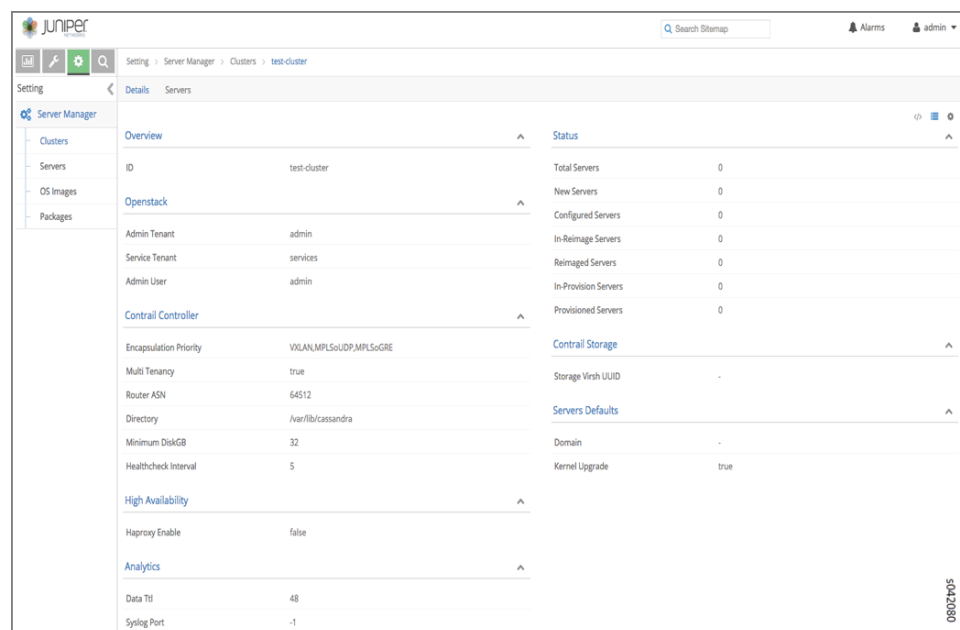
Click the upper right icon to switch to the JSON view to see the contents of the JSON file for the cluster, see [Figure 51 on page 215](#).

Figure 51: View Cluster JSON



The cluster name is a link, click the cluster name to display the cluster **Details** page, see [Figure 52 on page 215](#).

Figure 52: Link to View Cluster Details



Click the **Servers** tab to display the servers under that cluster, see [Figure 53 on page 216](#).

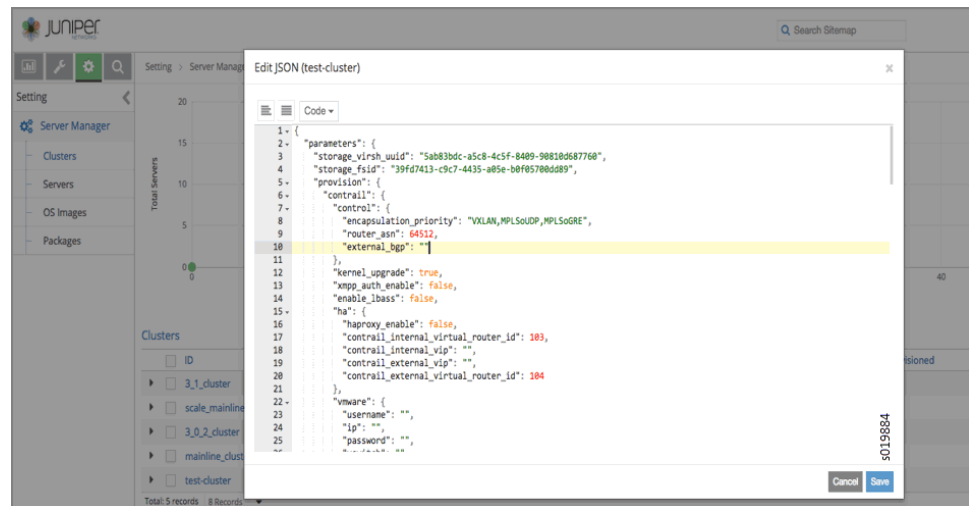
Figure 53: Display Servers for Cluster



Edit a Cluster through Edit JSON

Select **Edit JSON** to edit a cluster by editing the JSON file. Make changes to the JSON code and click **Save** to save the edited configuration for the cluster, see [Figure 54 on page 216](#).

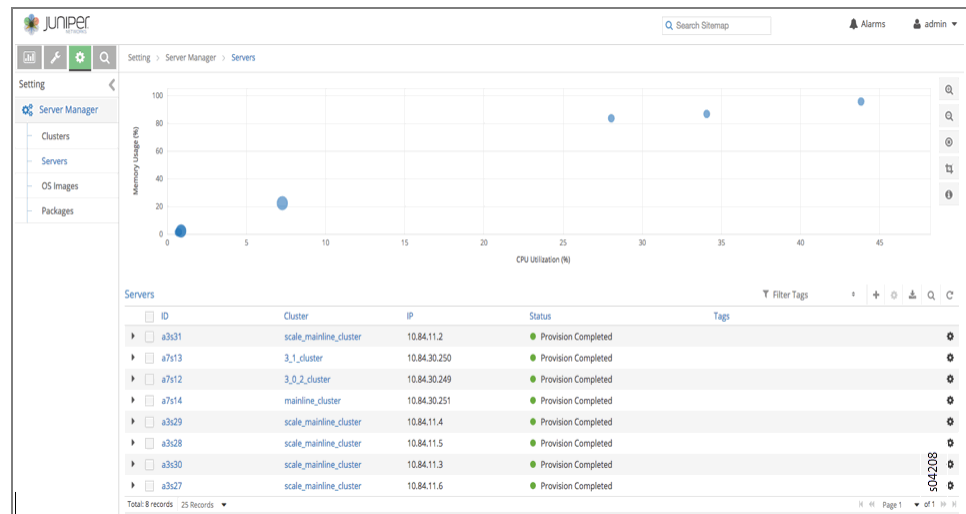
Figure 54: Edit Cluster JSON



Working with Servers in the Server Manager User Interface

Select **Setting > Server Manager** and click the **Servers** link in the left sidebar at to view a list of all servers, see [Figure 55 on page 217](#).

Figure 55: View Servers



Add a Server

To add a new server, select **Setting > Server Manager > Servers** and click the plus (+) icon at the upper right side in the header line. The **Add Server** window is displayed, see [Figure 56 on page 217](#), in which the **System Management** tab is expanded. Here you enter the details of **ID**, **Password**, **Domain**, **Partition**, and so on for the server.

Figure 56: Add Server, System Management

Add Server

System Management

ID: demo-server Password: [password field]

Host Name: demo-server Domain: englab.juniper.net

Static IP: [empty field] IPMI Address: 10.84.60.148

IPMI Username: ADMIN IPMI Password: [password field]

Partition: [empty field]

Interfaces

Cancel Save

In the following image, the **Physical Interfaces** icon is selected. You can add new interfaces or edit existing interfaces. To enable editing for any field, hover the cursor on any selected field to open it, see [Figure 57 on page 218](#).

Figure 57: Add Server, Physical Interfaces

Add Server

System Management

Physical Interfaces

Name	IP/Mask	MAC Address	Gateway	DHCP	TOR	TOR Port
eth01	1.2.3.4	aa:aa:aa:aa:aa:aa	2.2.3.4	<input checked="" type="checkbox"/>		

+Add

Bond Interfaces

OVS Type Switches

Contrail Storage

Provisioning

Cancel Save

3_1_cluster 10.84.30.250 Provision Completed

In the following image, the **Contrail Storage** icon is selected. You can configure parameters for Contrail Storage, including selecting a package and adding storage disks locations, see [Figure 58 on page 218](#).

Figure 58: Add Server, Contrail Storage

Add Server

System Management

Physical Interfaces

Bond Interfaces

OVS Type Switches

Contrail Storage

Storage Repo ID

Select Repo ID

Chassis ID

Select Chassis ID

Add New Chassis ID

Storage Disks

+Add

Provisioning

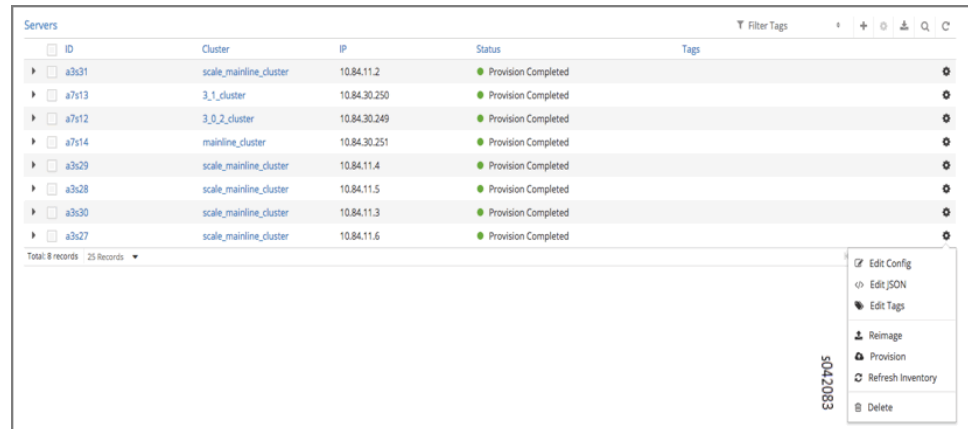
Cancel Save

scale_mainline_cluster 10.84.11.4 Provision Completed

When you are finished entering new server details in the **Add Server** window, click **Save** to add the new server configuration to the list of servers.

You can change details of the new server by clicking the gear wheel icon to the right side to get a list of actions available, including **Edit Config**, **Edit JSON**, **Edit Tags**, **Reimage**, **Provision**, **Refresh Inventory**, and **Delete**, see [Figure 59 on page 219](#).

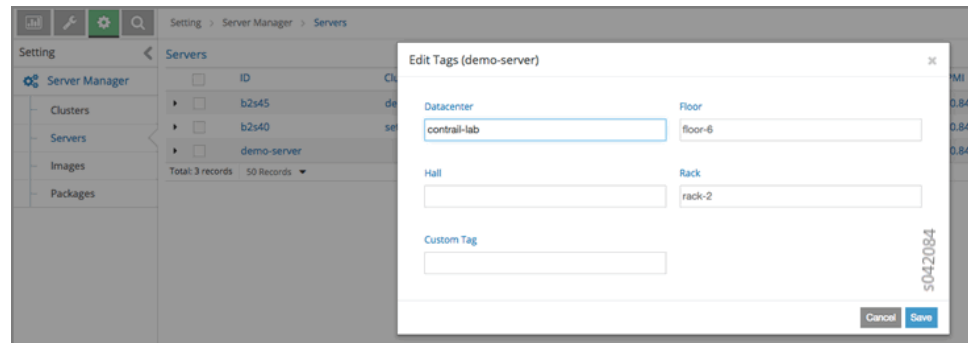
Figure 59: Select Server Actions



Edit Tags for Servers

Select **Edit Tags** from the gear wheel icon menu. The **Edit Tags** window is displayed. Enter any user-defined tags to be associated with the selected server, then click **Save** to add the tags to the server configuration, see [Figure 60 on page 219](#).

Figure 60: Edit Tags



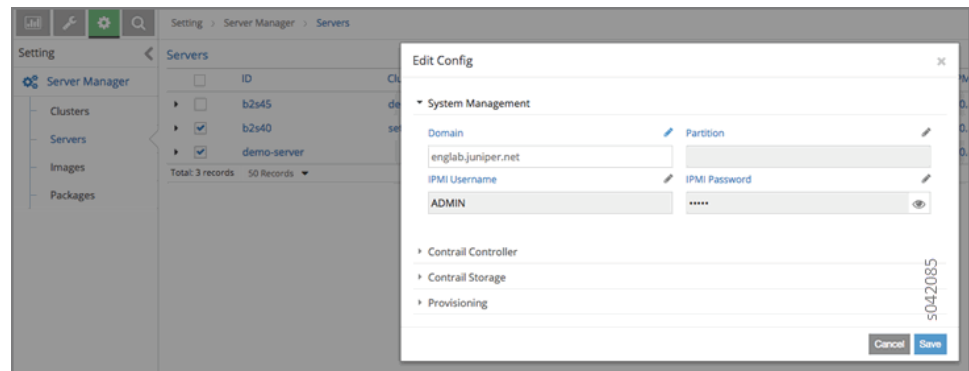
Using the Edit Config Option for Multiple Servers

You can also edit the configuration of multiple servers at one time. From the **Servers** window at **Setting > Server Manager > Servers**, select the servers you want to edit, then click a gear wheel icon at the right to open the action menu, and select **Edit Config**.

The **Edit Config** window is displayed, as shown.

Click a pencil icon to open configuration fields that can be edited. Fields include **System Management**, **Contrail Controller**, **Contrail Storage**, and so on, see [Figure 61 on page 220](#).

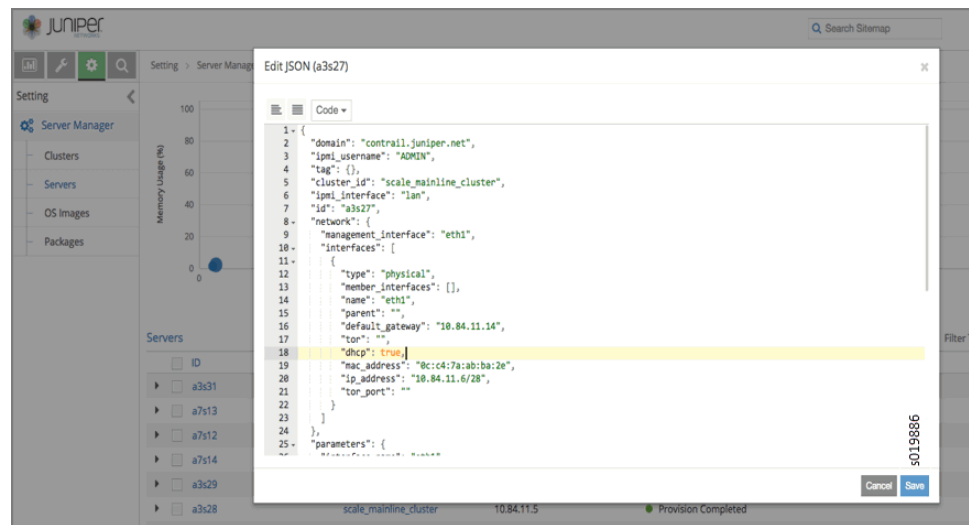
Figure 61: Edit Config, Multiple Servers



Edit a Server through Server Manager, Edit JSON

Select **Edit JSON** to edit the server through JSON file. Make changes to the server details in the JSON, then click **Save**, see [Figure 62 on page 220](#).

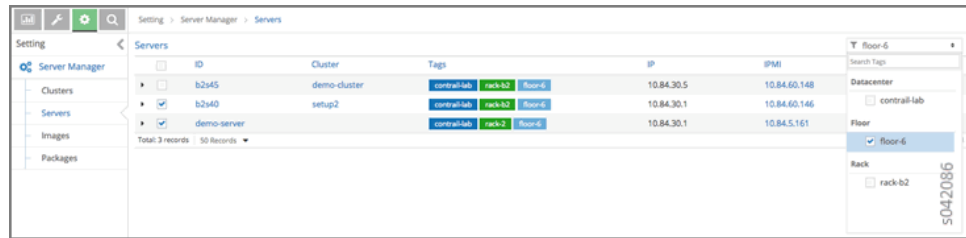
Figure 62: Server Edit JSON



Filter Servers by Tag

You can filter servers according to the tags defined for them. In the **Servers** window, click the **Filter Tags** field in the upper right heading. A list of configured tags is displayed. Select a tag by which to filter the list of servers, see [Figure 63 on page 221](#).

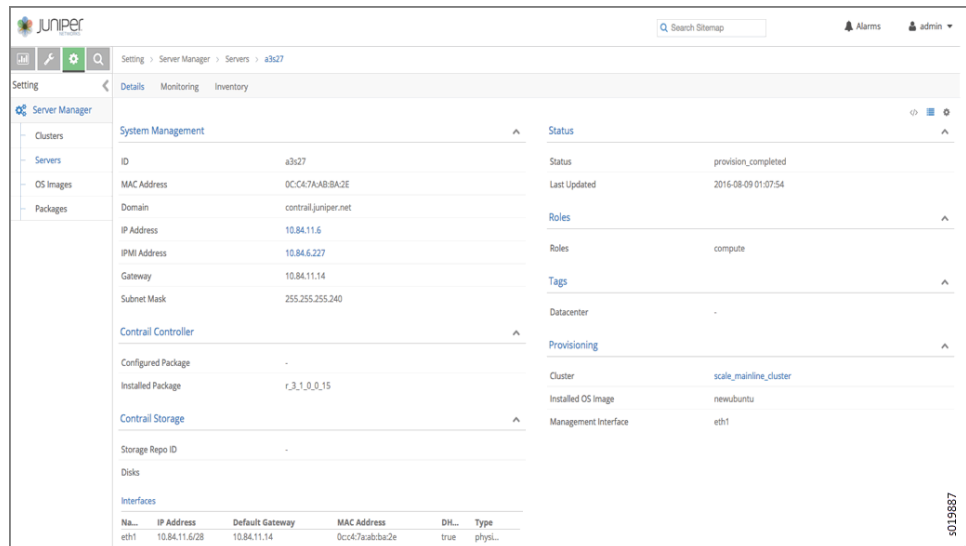
Figure 63: Filter Servers by Tag



Viewing Server Details

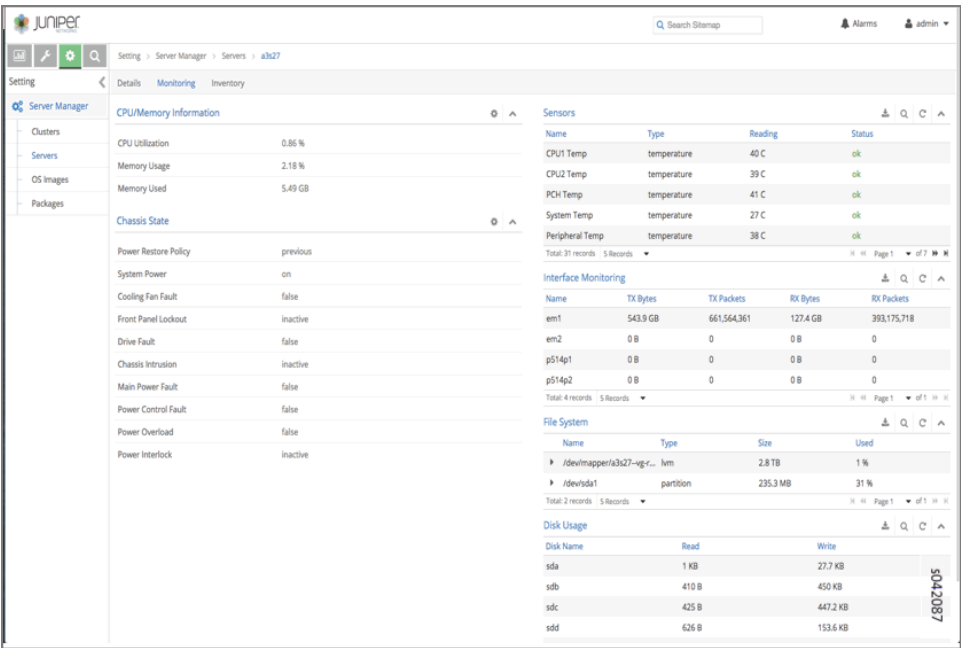
Each server name on the **Servers** page is a link to the details page for that server. Click any server name to open the details for that server, including **System Management** information, **Status**, **Contrail Controller**, **Contrail Storage**, **Roles**, **Tags**, and **Provisioning**, see Figure 64 on page 221.

Figure 64: View Server Details, System Management



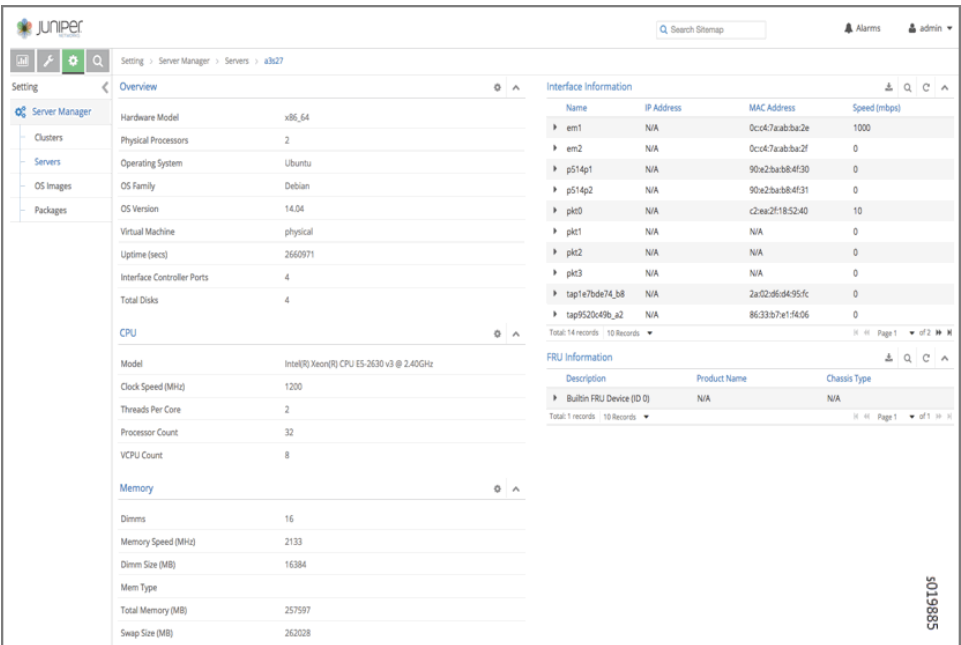
At the **Servers** page, click the **Monitoring** tab to see detailed information regarding **CPU/Memory Information**, **Chassis State**, **Sensors**, **Interface Monitoring**, **File System**, and **Disk Usage**, see Figure 65 on page 222.

Figure 65: Server Monitoring



At the **Servers** page, click the **Inventory** tab to see detailed information regarding **Overview** of the server, **Interface Information**, CPU information, Memory, and FRU Information, see [Figure 66 on page 222](#).

Figure 66: Server Inventory



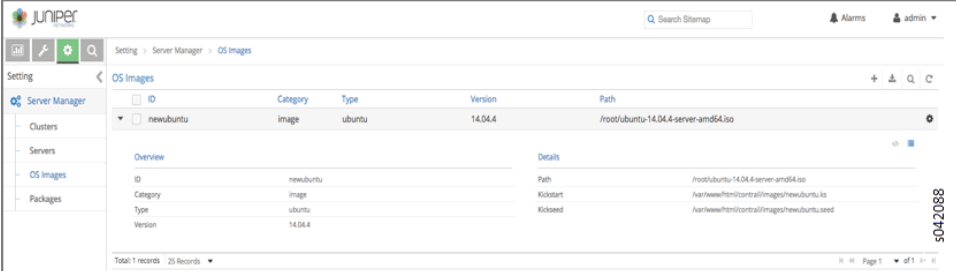
Configuring Images and Packages

Use the sidebar **Images and Packages** options to configure the software images and packages to be used by the Server Manager. Images are typically used to reimage clusters with an operating system version. Packages are used to provision clusters with a Contrail setup.

Both areas of the Server Manager user interface operate in a similar fashion. The figure shows the **Images** section. The **Packages** section has similar options.

Select **Images**. The Images page is displayed, see [Figure 67 on page 223](#).

Figure 67: Servers OS Images



ID	Category	Type	Version	Path
newubuntu	image	ubuntu	14.04.4	/root/ubuntu-14.04.4-server-amd64.iso

Overview		Details	
ID	newubuntu	Path	/root/ubuntu-14.04.4-server-amd64.iso
Category	image	Kickstart	/usr/share/contrail/images/newubuntu.iso
Type	ubuntu	Kickseed	/usr/share/contrail/images/newubuntu.seed
Version	14.04.4		

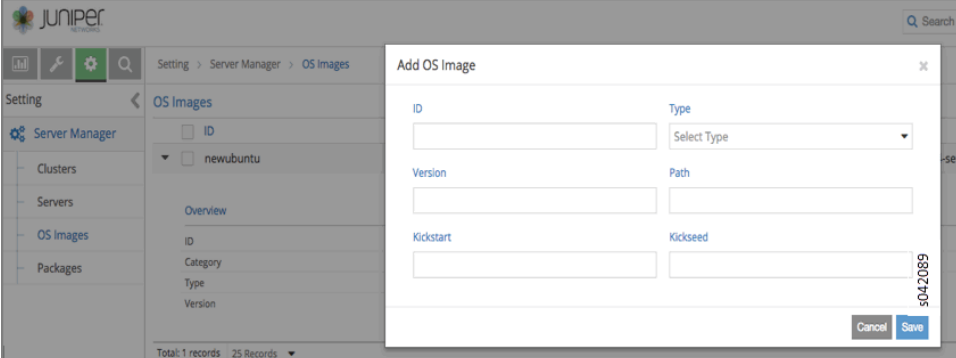
Add New Image or Package

To add a new image or package, on the respective **Images** or **Packages** page, click the plus (+) icon in the upper right header. The **Add Image** window is displayed. Enter the information for the new image (or package) and click **Save** to add the new item to the list of configured items, see [Figure 68 on page 223](#).



NOTE: The path field requires the path of the image where it is located on the server upon which the server-manager process is running.

Figure 68: Add OS Image



ID	Type
<input type="text"/>	Select Type

Version	Path
<input type="text"/>	<input type="text"/>

Kickstart	Kickseed
<input type="text"/>	<input type="text"/>

Cancel Save

Selecting Server Manager Actions for Clusters

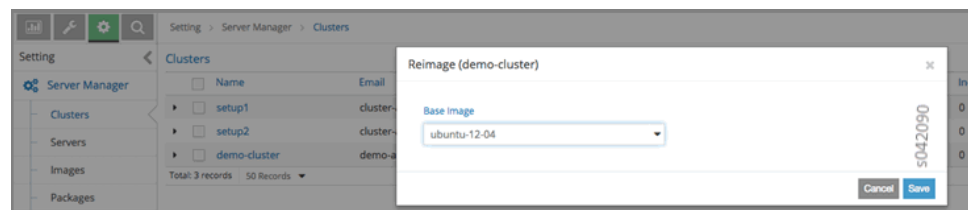
After all aspects of a cluster are configured, you can select actions for the Server Manager to perform on the cluster, such as **Reimage** or **Provision**.

Reimage a Cluster

Select **Setting > Servers > Clusters**. The **Clusters** window is displayed. Click the right side gear wheel icon of the cluster to be reimaged, then select **Reimage** from the action menu.

The **Reimage** dialog box is displayed, as shown. Verify that the correct image is selected in the **Default Image** field, then click **Save** to initiate the reimage action, see [Figure 69 on page 224](#).

Figure 69: Reimage Cluster

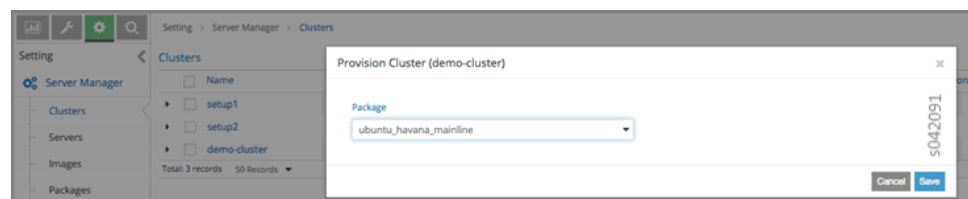


Provision a Cluster

The process to provision a cluster is similar to the process to reimage a cluster. Select **Setting > Servers > Clusters**. The **Clusters** window is displayed. Click the right side gear wheel icon of the cluster to be provisioned, then select **Provision** from the action menu.

The **Provision Cluster** dialog box is displayed, as shown. Verify that the correct package for provisioning is selected in the **Default Package** field, then click **Save** to initiate the provisioning action, see [Figure 70 on page 224](#).

Figure 70: Provision Cluster



- See Also**
- [Using Server Manager to Automate Provisioning on page 178](#)
 - [Installing Server Manager on page 173](#)
 - [Installing and Using Server Manager Lite on page 225](#)

Installing and Using Server Manager Lite

This topic describes how to install and troubleshoot Server Manager Lite.

Server Manager Lite Overview

Server Manager Lite (SM-Lite), is a streamlined version of the Server Manager software that does not include the reimage function.

SM-Lite supports the Server Manager functions of provisioning, monitoring, inventory, and WebUI. SM-Lite is a replacement for the fab provisioning used in previous versions. SM-Lite allows easy deployment of Contrail provisioning and enables developers to work in isolated environments for Contrail provisioning.

SM-Lite eliminates installation and configuration of DHCP, DNS, and Cobbler services. Additionally, SM-Lite installation setup scripts are enhanced to reduce installation time.

SM-Lite provides a single command to install SM-Lite and provision a Contrail cluster.

SM-Lite introduces additional capabilities into Server Manager. The SM-Lite package is part of the Contrail Server Manager installer Debian package (`contrail-server-manager-installer_<version string>.deb`).

SM-Lite works with or without having a separate node for the SM-Lite installation, it can be installed on any Contrail node, but it is recommended to install it on the config node.

SM-Lite uses the Server Manager Web UI functionality and it can run on the same node as the Contrail WebUI. Consequently, the default port for the Server Manager Web UI is **9080**.

Server Manager and SM-Lite share some code base for provisioning, so changes or enhancements made to Server Manager provisioning functionality are also available in SM-Lite. SM-Lite can be used to deploy Contrail networking, in addition to deploying Contrail Cloud.

Installing Server Manager Lite

The SM-Lite package is included as part of the Server Manager installer package.

The installer package also has other packages such as Server Manager, Server Manager client, Server Manager WebUI, and Server Manager inventory. Before provisioning commands can be executed using SM-Lite, you need to install the Server Manager installer package.

Use the following command to install the Server Manager installer package.

```
dpkg -i <contrail-server-manager-installer-deb>
```

After the Server Manager installer package is installed, all necessary Server Manager packages, scripts, and so on are made available on the server where it is installed. You can then start using Server Manager Lite commands.

Provisioning Using SM-Lite with Contrail 4.0

For Contrail 4.0, to provision the target systems, use the **provision_containers.sh** script.

The **provision_containers.sh** script can be run either by specifying cluster, server, and image definition JSON files, or by specifying all of the provisioning parameters in a **testbed.py** file (as used previously for provisioning with fab in earlier versions of Contrail software). The Contrail package provided to the script can be either a Contrail networking package or a Contrail cloud package for the base OS platform running on the servers.

- When using JSON files:

```
/opt/contrail/contrail_server_manager/provision_containers.sh --ij  
<image_json_path> --cj <cluster_json_path> --sj <server_json_path> --cluster-id  
<cluster-id>
```

- When using **testbed.py** file:

```
/opt/contrail/contrail_server_manager/provision_containers.sh --testbed <testbed>  
--contrail-package <contrail-package> --cluster-id <cluster-id>
```

The **provision_containers.sh** script performs the following functions:

1. Installs SM-Lite.

Uses the **setup.sh** installation script with the **-smlite** option to install the SM-Lite package, (**contrail-server-manager-lite_<version-sku>_all.deb**) and all other needed packages on the system.

2. Prepares the cluster for Contrail provisioning.

Translates the parameters in the **testbed.py** file or the JSONs provided into Server Manager objects and stores them in the Server Manager database. This specifies the servers in the cluster and the configuration parameters. The cluster-id value is used, if it is specified.

3. Performs a pre-check on the target systems to ensure that they are ready for running provisioning. SM-Lite uses Ansible playbooks and Docker containers from the Contrail Release 4.0 package to provision the Contrail cluster. It also uses Puppet modules and manifests from the Contrail package to provision Openstack.

4. This step issues provisioning commands for the cluster with the given Contrail package.

Server Manager Lite can be installed on any node. We recommend that you install it on the config or controller node. Server Manager Lite can be installed on a separate node other than the Contrail cluster nodes.

The Server Manager WebUI default port is **9080**. You can change the port by editing the **/etc/contrail/config.global.sm.js** file, and then restarting the **supervisor-webui-sm** process.

Displaying the Cluster Status

The **server-manager display cluster -detail** command displays the provisioning status of a cluster by role and by role progress.

Use the **server-manager status server** command to display the current status of the servers.

Use the **server-manager status provision** command to display the current status of the provision under progress.

Displaying the SM-Lite Installation and Provisioning Log Files

Log files that provide information during installation and use of SM-Lite software are available at:

- `/var/log/contrail/install_logs/install_<timestamp>.log` (SM-Lite install)
- `/var/log/contrail/install_logs/provision_<timestamp>.log` (provisioning command logs)
- `testbed_parser.log` and `preconfig.log`

Contrail Provisioning Log Files

For each Puppet run, log files are automatically uploaded to the Server Manager at the following locations:

- `http:<sm-lite-ip-address>/logs`
- `/var/log/contrail_server_manager/<target>/<timestamp>.log`
- `/var/log/contrail/*`

You can also display the status of the processes and services using the **contrail-status** command.

Related Documentation

- [Using Server Manager to Automate Provisioning on page 178](#)
- [Using the Server Manager Web User Interface on page 206](#)
- [Installing Server Manager on page 173](#)

CHAPTER 9

Extending Contrail to Physical Routers, Bare Metal Servers, Switches, and Interfaces

- [Using ToR Switches and OVSDb to Extend the Contrail Cluster to Other Instances on page 229](#)
- [Configuring High Availability for the Contrail OVSDb ToR Agent on page 239](#)
- [Using Device Manager to Manage Physical Routers on page 244](#)
- [SR-IOV VF as the Physical Interface of vRouter on page 269](#)
- [Using Gateway Mode to Support Remote Instances on page 270](#)
- [REST APIs for Extending the Contrail Cluster to Physical Routers, and Physical and Logical Interfaces on page 273](#)

Using ToR Switches and OVSDb to Extend the Contrail Cluster to Other Instances

- [Support for ToR Switch and OVSDb Overview on page 229](#)
- [ToR Services Node \(TSN\) on page 230](#)
- [Contrail ToR Agent on page 230](#)
- [Using the Web Interface to Configure ToR Switch and Interfaces on page 232](#)
- [Configuration Parameters for Provisioning ToR and TSN on page 234](#)
- [Prerequisite Configuration for QFX5100 Series Switch on page 237](#)
- [Changes to Agent Configuration File on page 238](#)
- [REST APIs on page 239](#)

Support for ToR Switch and OVSDb Overview

Contrail Releases 2.1 and greater support extending a cluster to include bare metal servers and other virtual instances connected to a top-of-rack (ToR) switch that supports the Open vSwitch Database Management (OVSDb) protocol. The bare metal servers and other virtual instances can belong to any of the virtual networks configured in the Contrail cluster, facilitating communication with the virtual instances running in the cluster. Contrail policy configurations can be used to control this communication.

The OVSDB protocol is used to configure the ToR switch and to import dynamically-learned addresses. VXLAN encapsulation is used in the data plane communication with the ToR switch.

ToR Services Node (TSN)

A ToR services node (TSN) can be provisioned as a role in the Contrail system. The TSN acts as the multicast controller for the ToR switches. The TSN also provides DHCP and DNS services to the bare metal servers or virtual instances running behind ToR switch ports.

The TSN receives all the broadcast packets from the ToR switch, and replicates them to the required compute nodes in the cluster and to other EVPN nodes. Broadcast packets from the virtual machines in the cluster are sent directly from the respective compute nodes to the ToR switch.

The TSN can also act as the DHCP server for the bare metal servers or virtual instances, leasing IP addresses to them, along with other DHCP options configured in the system. The TSN also provides a DNS service for the bare metal servers. Multiple TSN nodes can be configured in the system based on the scaling needs of the cluster.

Contrail ToR Agent

A ToR agent provisioned in the Contrail cluster acts as the OVSDB client for the ToR switch, and all of the OVSDB interactions with the ToR switch are performed by using the ToR agent. The ToR agent programs the different OVSDB tables onto the ToR switch and receives the local unicast table entries from the ToR switch.

The ToR agent receives the configuration information for the ToR switch, translates the Contrail configuration to OVSDB, and populates the relevant OVSDB table entries in the ToR switch.

Contrail recognizes the ToR after you configure **tsn** and **toragent** roles.

The typical practice is to run the ToR agent on the TSN node.

Configuration Model

[Figure 71 on page 231](#) depicts the configuration model used in the system.

Figure 71: Configuration Model

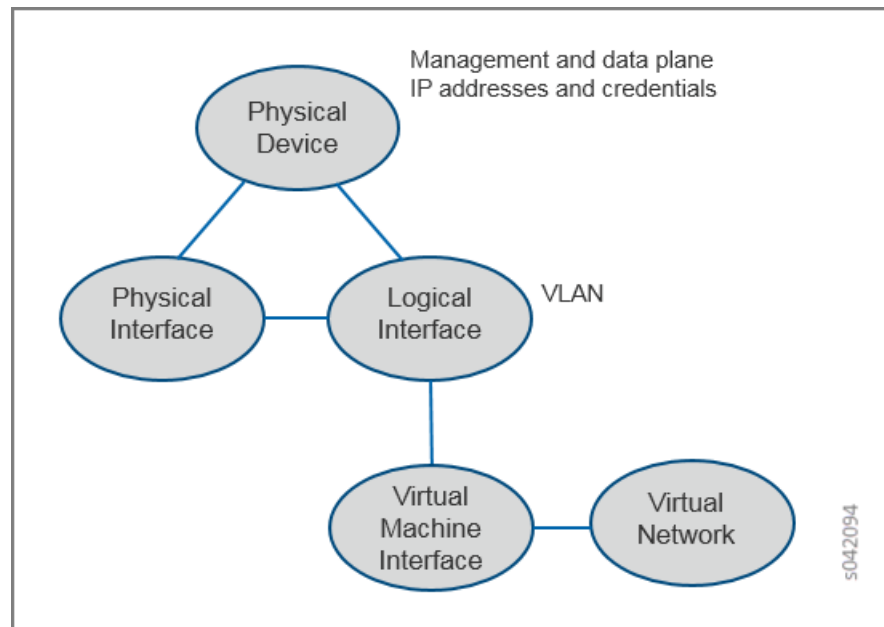


Table 28 on page 231 maps the Contrail configuration objects to the OVSDb tables.

Table 28: Contrail Objects in the OVSDb

Contrail Object	OVSDb Table
Physical device	Physical switch
Physical interface	Physical port
Logical interface	<VLAN physical port> binding to logical switch
Virtual networks	Logical switch
Layer 2 unicast route table	Unicast remote and local table
	Multicast remote table
	Multicast local table
	Physical locator table
	Physical locator set table

Control Plane

The ToR agent receives the EVPN route entries for the virtual networks in which the ToR switch ports are members, and adds the entries to the unicast remote table in the OVSDb.

MAC addresses learned in the ToR switch for different logical switches (entries from the local table in OVSDB) are propagated to the ToR agent. The ToR agent exports the addresses to the control node in the corresponding EVPN tables, which are further distributed to other controllers and subsequently to compute nodes and other EVPN nodes in the cluster.

The TSN node receives the replication tree for each virtual network from the control node. It adds the required ToR addresses to the received replication tree, forming its complete replication tree. The other compute nodes receive the replication tree from the control node, whose tree includes the TSN node.

Data Plane

The data plane encapsulation method is VXLAN. The virtual tunnel endpoint (VTEP) for the bare metal end is on the ToR switch.

Unicast traffic from bare metal servers is VXLAN-encapsulated by the ToR switch and forwarded, if the destination MAC address is known within the virtual switch.

Unicast traffic from the virtual instances in the Contrail cluster is forwarded to the ToR switch, where VXLAN is terminated and the packet is forwarded to the bare metal server.

Broadcast traffic from bare metal servers is received by the TSN node. The TSN node uses the replication tree to flood the broadcast packets in the virtual network.

Broadcast traffic from the virtual instances in the Contrail cluster is sent to the TSN node, which replicates the packets to the ToR switches.

Using the Web Interface to Configure ToR Switch and Interfaces

The Contrail Web user interface can be used to configure a ToR switch and the interfaces on the switch. To add a switch, select **Configure > Physical Devices > Physical Routers**.

The **Physical Routers** list is displayed.

Click the + symbol to open the **Add** menu. From the **Add** menu you can select one of the following:

- **Add OVSDB Managed ToR**
- **Add Netconf Managed Physical Router**
- **CPE Router**
- **Physical Router**

To add a physical ToR, select **Add OVSDB Managed ToR**. The **Create** window is displayed, as shown in [Figure 72 on page 233](#). Enter the IP address and VTEP address of the ToR switch. Also configure the TSN and ToR agent names for the ToR.

Figure 72: Create OVSDb Managed ToR

The screenshot shows a 'Create' window with a close button (X) in the top right corner. The window has two tabs: 'OVSDb Managed ToR' (selected) and 'Permissions'. The form contains the following fields:

- Name:** A text input field.
- Vendor:** A text input field.
- Model:** A text input field.
- Management IP:** A text input field.
- VTEP Address:** A text input field.
- TOR Agent(s):** Two dropdown menus, each with the placeholder text 'Select or Enter TOR Agent Name'.
- TSN(s):** Two dropdown menus, each with the placeholder text 'Select or Enter TSN Name'.
- SNMP Monitored:** A checkbox.

At the bottom right, there is a 'Cancel' button and a 'Save' button. A vertical text label 's042093' is positioned on the right side of the window.

To add the logical interfaces to be configured on the ToR switch, select **Configure > Physical Devices > Interfaces**.

The **Physical Routers** list is displayed. Click the + symbol. The **Add Interface** window is displayed, as shown in [Figure 73 on page 234](#).

At **Add Interface**, enter the name of the logical interface. The name must match the name on the ToR, for example, ge-0/0/0.10. Also enter other logical interface configuration parameters, such as VLAN ID, MAC address, and IP address of the bare metal server and the virtual network to which it belongs.

Figure 73: Add Interface

Configuration Parameters for Provisioning ToR and TSN

This section presents the configuration parameters for different methods of provisioning ToR and TSN.

- [Inventory Format ToR and TSN on page 235](#)
- [JSON Format ToR and TSN on page 235](#)
- [Testbed.py Format ToR and TSN on page 236](#)

The following information can be provided for each ToR agent.

- IP address of the ToR
- a unique numeric identifier for the ToR
- a unique (optional) name for the ToR Agent
- the OVS protocol (TCP or SSL)
 - the OVS port
 - when OVS protocol is TCP, port indicates the TCP port to connect on the ToR
 - when OVS protocol is pssl, port indicates the SSL port on which the ToR agent listens for connections from the TOR
- TSN IP address of the ToR
- name of the TSN node
- IP address of the data tunnel endpoint

- HTTP server port of the ToR Agent using which introspect data can be checked
- vendor name for ToR (optional)
- product name of ToR switch (optional)
- OVS keepalive timeout (optional)

Inventory Format ToR and TSN

Indicate the compute node to act as TSN.

```
[contrail-computes]
1.1.1.7 ctrl_data_ip=10.1.1.7 tsn_mode=True

tor_agent = { 'host1': [
    { 'tor_ip': '10.xxx.221.35',
      'tor_agent_id': '1',
      'tor_agent_name': 'node-1',
      'tor_ovs_protocol': 'tcp',
      'tor_ovs_port': '9999',
      'tor_tsn_ip': '10.xxx.221.33',
      'tor_tsn_name': 'tsn1',
      'tor_name': 'contrail-tor-1',
      'tor_tunnel_ip': '5.5.5.5',
      'tor_http_server_port': '9090',
      'tor_vendor_name': 'Juniper',
      'tor_product_name': 'QFX5100',
      'tor_agent_ovs_ka': '1000'
    },
    { ... }
  ],
  'host2': [ ... ]
}
```

JSON Format ToR and TSN

If you are provisioning using Server Manager or SM-Lite, you can provision with JSON. The following is the JSON format.

For ToR in server.json.

```
{
  "server": [
    {
      "id": "new-server",
      "parameters": {
        "top_of_rack": {
          "switches": [
            {
              "agent_id": "1",
              "ip": "10.x.141.84",
              "tunnel_ip": "10.xx.141.84",
              "name": "TOR1",
              "tsn_name": "TSN1",
              "agent_name": "AGENT1",
              "ovs_port": "6632",
              "agent_ovs_ka": "1000",
              "ovs_protocol": "tcp",
            }
          ]
        }
      }
    }
  ]
}
```

```

        "http_server_port": "9912",
        "vendor_name": "Juniper"
    },
    {
        "agent_id": "2",
        "ip": "10.xx.141.83",
        "tunnel_ip": "10.xx.141.83",
        "name": "TOR2",
        "ovs_port": "6632",
        "ovs_protocol": "tcp",
        "http_server_port": "9913",
        "vendor_name": "Juniper"
    }
]
},

```

For TSN in server.json.

```

{
  "server" : [
    {
      "id": "new-server",
      "parameters" : {
        "provision": {
          "contrail_4": {
            "tsn_mode": false
          }
        }
      }
    }
  ]
}

```

Testbed.py Format ToR and TSN

Starting with Contrail 4.0, if you are provisioning using SM-Lite, you can provision with JSON or testbed.py. The following is the testbed.py format.

The ToR agent and TSN can be provisioned using the **testbed.py** configured with the following:

- The **env.roledef** section is configured with the **tsn** and **toragent** roles. The hosts for these roles should also host a compute node.
- The **env.tor_agent** section should be present and configured.

For ToR:

```

#env.tor_agent = {host10:[{
#           'tor_ip': '10.xxx.217.39',
#           'tor_agent_id': '1',
#           'tor_agent_name': 'nodexx-1',
#           'tor_type': 'ovs',
#           'tor_ovs_port': '9999',
#           'tor_ovs_protocol': 'tcp',
#           'tor_tsn_name': 'nodec45',
#           'tor_name': 'bng-contrail-qfx51-2',
#           'tor_tunnel_ip': '34.34.34.34',
#           'tor_vendor_name': 'Juniper',
#           'tor_product_name': 'QFX5100',
#           'tor_agent_http_server_port': '9010',

```

```
#           'tor_agent_ovs_ka': '10000',
#           }
#       }
```

For TSN:

```
env.roledefs = {
  'tsn': [host1], # Optional, Only to enable TSN. Only compute can support
  TSN
}
```

For more information, see

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

Prerequisite Configuration for QFX5100 Series Switch

When using the Juniper Networks QFX5100 Series switches, ensure the following configurations are made on the switch before extending the Contrail cluster.

1. Enable OVSDDB.
2. Set the connection protocol.
3. Identify the interfaces that are managed by means of OVSDDB.
4. Configure the controller (in case pssl is used). If HA Proxy is used, use the address of the HA Proxy node and use the VIP when VRRP is used between multiple nodes running HA Proxy. The following is an example:

```
set interfaces lo0 unit 0 family inet address
set switch-options ovssdb-managed
set switch-options vtep-source-interface lo0.0
set protocols ovssdb interfaces
set protocols ovssdb passive-connection protocol tcp port
set protocols ovssdb controller <tor-agent-ip> inactivity-probe-duration 10000
protocol ssl port <tor-agent-port>
```

5. When using SSL to connect, CA-signed certificates must be copied to the `/var/db/certs` directory in the QFX device. The following example shows one way to get the certificates. The following commands could be run on any server.

```
apt-get install openvswitch-common
ovs-pki init
ovs-pki req+sign vtep
scp vtep-cert.pem root@<qfx>:/var/db/certs
scp vtep-privkey.pem root@<qfx>:/var/db/certs
cacert.pem file will be available in /var/lib/openvswitch/pki/switchca, when
the above are done. This is the file to be provided in the above testbed (in
env.ca_cert_file).
```

Debug QFX5100 Configuration

You can use the following commands on the QFX switch to show the OVSDb configuration.

```
show ovssdb logical-switch
```

```
show ovssdb interface
```

```
show ovssdb mac
```

```
show ovssdb controller
```

```
show vlans
```

You can use the agent introspect on the ToR agent and the TSN nodes to show the configuration and operational state of these modules.

- The TSN module is like any other **contrail-vrouter-agent** on a compute node, with introspect access available on port 8085 by default. Use the introspect on port 8085 to view operational data such as interfaces, virtual network, and VRF information, along with their routes.
- The port on which the ToR agent introspect access is available is in the configuration file provided to the contrail-tor-agent. This provides the OVSDb data available through the client interface, apart from the other data available in a Contrail Agent.

Changes to Agent Configuration File

You can make changes to the agent features by making changes in the configuration file.

In the `/etc/contrail/contrail-vrouter-agent.conf` file for TSN, the **agent_mode** option is available in the DEBUG section to configure the agent to be in TSN mode.

```
agent_mode = tsn
```

The following are typical configuration items in a ToR agent configuration file.

```
[DEFAULT]
```

```
agent_name = noded2-1 # Name (formed with hostname and TOR id from below)
```

```
agent_mode = tor # Agent mode
```

```
http_server_port=9010 # Port on which Introspect access is available
```

```
[TOR]
```

```
tor_ip=<ip> # IP address of the TOR to manage
```

```
tor_id=1 # Identifier for ToR Agent.
```

```
tor_type=ovs # ToR management scheme - only "ovs" is supported
```

```
tor_ovs_protocol=tcp # IP-Transport protocol used to connect to TOR, can be tcp
```

```
or pssl

tor_ovs_port=port # OVS server port number on the ToR

tsn_ip=<ip> # IP address of the TSN

tor_heartbeat_interval=10000 # heartbeat timer in ms

ssl_cert=/etc/contrail/ssl/certs/tor.1.cert.pem # path to SSL certificate on TOR
Agent, needed for pssl

ssl_privkey=/etc/contrail/ssl/private/tor.1.privkey.pem # path to SSL private key
on TOR Agent, needed for pssl

ssl_cacert=/etc/contrail/ssl/certs/cacert.pem # path to SSL CA cert on the node,
needed for pssl
```

REST APIs

For information regarding REST APIs for physical routers and physical and logical interfaces, see [“REST APIs for Extending the Contrail Cluster to Physical Routers, and Physical and Logical Interfaces”](#) on page 273.

Related Documentation

- [REST APIs for Extending the Contrail Cluster to Physical Routers, and Physical and Logical Interfaces](#) on page 273
- [Using Device Manager to Manage Physical Routers](#) on page 244
- [Configuring High Availability for the Contrail OVSDb ToR Agent](#) on page 239

Configuring High Availability for the Contrail OVSDb ToR Agent

This topic describes how high availability can be configured for the Contrail ToR agent.

- [Overview: High Availability for a ToR Switch](#) on page 239
- [High Availability Solution for Contrail ToR Agent](#) on page 240
- [Failover Methodology Description](#) on page 241
- [Failure Scenarios](#) on page 241
- [Redundancy for HAProxy](#) on page 243
- [Configuration for ToR Agent High Availability](#) on page 243

Overview: High Availability for a ToR Switch

In Contrail Release 2.20 and later, high availability can be configured for the Contrail ToR agent.

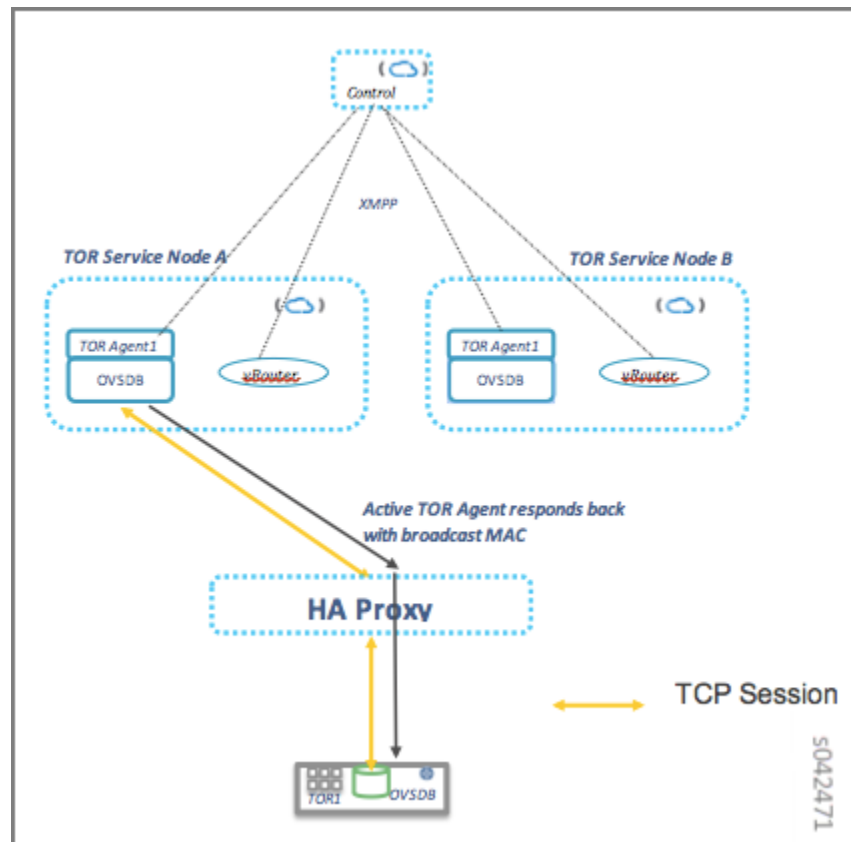
When a top-of-rack (ToR) switch is managed through the Open vSwitch Database (OVSDb) protocol by using a ToR agent on Contrail, a high availability configuration is necessary to maintain ToR agent redundancy. With ToR agent redundancy, if the ToR agent responsible for a ToR switch is unable to act as the vRouter agent for the ToR switch, due to any failure condition in the network or the node, then another ToR agent takes over and manages the ToR switch.

ToR agent redundancy (high availability) is achieved using HAProxy. HAProxy is an open source, reliable solution that offers high availability and proxy service for TCP applications. The solution uses HAProxy to initiate an SSL connection from the ToR switch to the ToR agent. This configuration ensures that the ToR switch is connected to exactly one active ToR agent at any given point in time.

High Availability Solution for Contrail ToR Agent

The following figure illustrates the method for achieving high availability for the ToR agent in Contrail.

Figure 74: High Availability Solution for Contrail ToR Agent



The following describes the events shown in the figure:

- ToR agent redundancy is achieved using HAProxy.
- Two ToR agents are provisioned on different TSN nodes, to manage the same ToR switch.
- Both ToR agents created in the cluster are active and get the same information from the control node.
- HAProxy monitors these ToR agents.
- An SSL connection is established from the ToR switch to the ToR agent, via HAProxy.

- HAProxy selects one ToR agent to establish the SSL connection (e.g., ToR Agent 1 running on TSN A).
- Upon connection establishment, this ToR Agent adds the ff:ff:ff:ff:ff:ff broadcast MAC address in the OVSDB with its own TSN IP address.
- The ToR Agent sends the MAC addresses of the bare metal servers learned by the ToR switch to the control node using XMPP.
- The control node reflects the addresses to other ToR agents and vRouter agents.

Failover Methodology Description

The ToR switch connects to the HAProxy that is configured to use one of the ToR agents on the two ToR services nodes (TSNs). An SSL connection is established from the ToR switch to the ToR agent, making that agent the active ToR agent. The active ToR agent is responsible for managing the OVSDB on the ToR switch. It configures the OVSDB tables based on the configuration. It advertises the MAC routes learned on the ToR switch as Ethernet VPN (EVPN) routes to the Contrail controller. It also programs any routes learned by means of EVPN over XMPP, southbound into OVSDB on the ToR switch.

The active ToR agent also advertises the multicast route (ff:ff:ff:ff:ff:ff) to the ToR switch, ensuring that there is only one multicast route in OVSDB pointing to the active TSN.

Both the ToR agents, active and standby, receive the same configuration from the control node, and all routes are synchronized by means of BGP.

After the SSL connection is established, keepalive messages are exchanged between the ToR switch and the ToR agent. The messages can be sent from either end and are responded to from the other end. When any message exchange is seen on the connection, the keepalive message is skipped for that interval. When the ToR switch sees that keepalive has failed, it closes the current SSL session and attempts to reconnect. When the ToR agent side sees that keepalive has failed, it closes the SSL session and retracts the routes it exported to the control node.

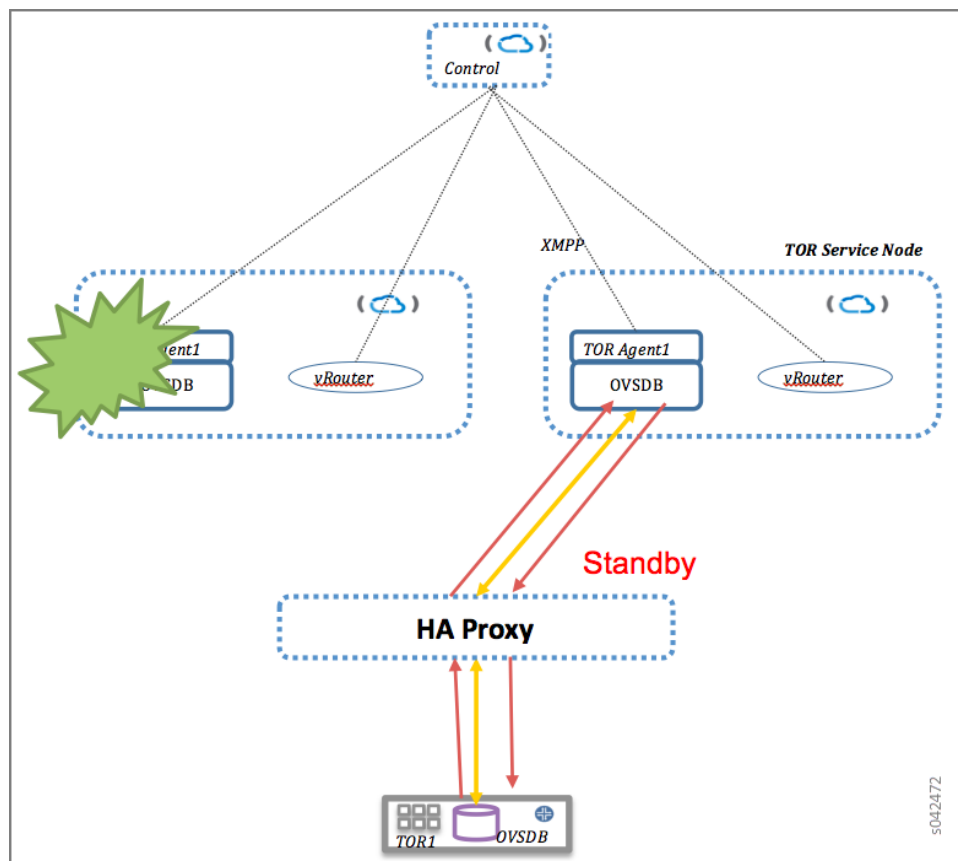
Failure Scenarios

Whenever the HAProxy cannot communicate with the ToR agent, a new SSL connection from the ToR switch is established to the other ToR agent.

HAProxy communication failures can occur under several scenarios, including:

- The node on which the ToR agent is running goes down or fails.
- The ToR agent crashes.
- A network or other issue prevents or interrupts HAProxy communication with the ToR agent.

Figure 75: Failure Scenarios



When a connection is established to the other ToR agent, the new ToR agent does the following:

- Updates the multicast route in OVSDB to point to the new TSN.
- Gets all of the OVSDB entries.
- Audits the data with the configurations available.
- Updates the database.
- Exports entries from the OVSDB local table to the control node.

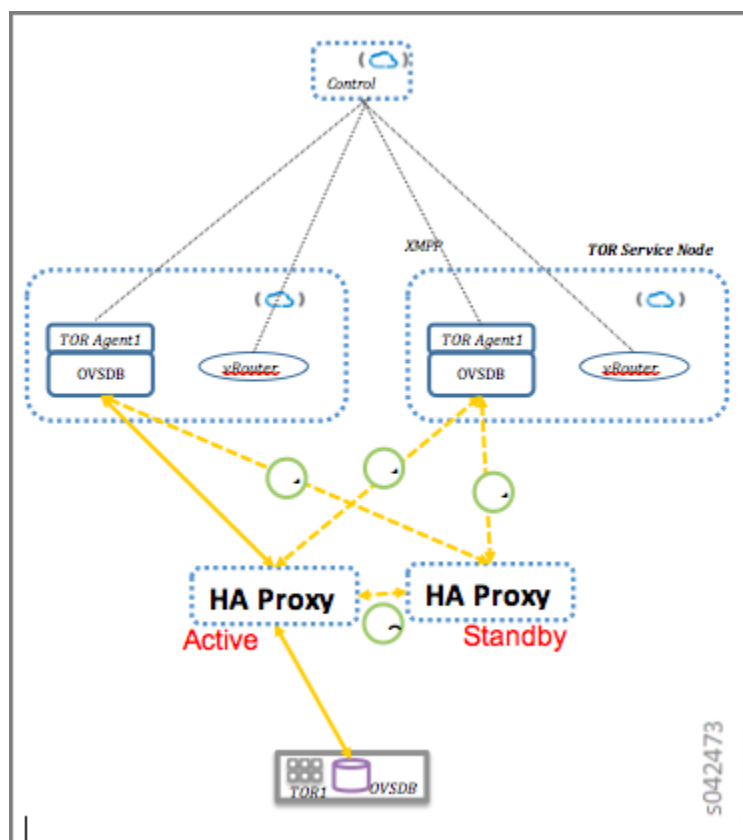
Because the configuration and routes from the control node are already synchronized to the new ToR Services Node (TSN), the new TSN can immediately act on the broadcast traffic from the ToR switch. Any impact to the service is only for the time needed for the SSL connection to be set up and for programming the multicast and unicast routes in the OVSDB.

When the SSL connection goes down, the ToR agent retracts the routes exported. Also, if the Extensible Messaging and Presence Protocol (XMPP) connection between the ToR agent and the control node goes down, the control node removes the routes exported by the ToR agent. In these scenarios, the entries from the OVSDB local table are retracted and then added back from the new ToR agent.

Redundancy for HAProxy

In a high availability configuration, multiple HAProxy nodes are configured, with Virtual Router Redundancy Protocol (VRRP) running between them. The ToR agents are configured to use the virtual IP address of the HAProxy nodes to make the SSL connection to the controller. The active TCP connections go to the virtual IP master node, which proxies them to the chosen ToR agent. A ToR agent is chosen based on the number of connections from the HA Proxy to that node (the node with lower number of connections gets the new connection) and can be controlled through configuration of the HAProxy.

Figure 76: Redundancy for HAProxy



If the HAProxy node fails, a standby node becomes the virtual IP master and sets up the connections to the ToR agents. The SSL connections are reestablished, following the same methods discussed earlier.

Configuration for ToR Agent High Availability

To get the required configuration downloaded from the control node to the TSN agent and to the ToR agent, the physical router node must be linked to the virtual router nodes that represent the two ToR agents and the two TSNs.

In the Contrail Web user interface select **Configure > Physical Devices > Physical Routers**. In the **Physical Routers** window, click the + icon. The **Add OVSDDB Managed ToR** window is displayed. See [Figure 77 on page 244](#).

Figure 77: Add OVSDDB Managed ToR Window

The screenshot shows the 'Add OVSDDB Managed ToR' configuration window. It includes the following fields and controls:

- Name:** A text input field.
- Vendor:** A text input field.
- Model:** A text input field.
- Management IP:** A text input field.
- VTEP Address:** A text input field.
- TOR Agent(s):** Two dropdown menus, each with the placeholder text 'Select or Enter Name'.
- TSN(s):** Two dropdown menus, each with the placeholder text 'Select or Enter Name'.
- SNMP Monitored:** A checked checkbox.
- SNMP Settings:** A section header with a dropdown arrow.
- Version:** Radio buttons for '2' (selected) and '3'.
- Buttons:** 'Cancel' and 'Save' buttons at the bottom right.

Enter a name to create an entry for the ToR switch, enter the ToR switch management IP address, and enter the virtual tunnel endpoint (VTEP) address. The router name should match the hostname of the ToR switch. Both ToR agents and their respective TSN nodes can be configured here.

- Related Documentation**
- [Using ToR Switches and OVSDDB to Extend the Contrail Cluster to Other Instances on page 229](#)

Using Device Manager to Manage Physical Routers

- [Support for Physical Routers Overview on page 245](#)
- [Configuration Model on page 245](#)
- [Alternate Ways to Configure a Physical Router on page 247](#)
- [Device Manager Configurations on page 247](#)
- [Prerequisite Configuration Required on MX Series Device on page 248](#)
- [Configuration Scenarios on page 248](#)
- [Device Manager Functionality on page 248](#)
- [Dynamic Tunnels on page 249](#)

- [Extending the Public Network on page 255](#)
- [Ethernet VPN Configuration on page 256](#)
- [Floating IP Addresses and Source Network Address Translation for Guest Virtual Machines and Bare Metal Servers on page 257](#)
- [Samples of Generated Configurations for an MX Series Device on page 265](#)

Support for Physical Routers Overview

A configuration node daemon named Device Manager can be used to manage physical routers in the Contrail system.

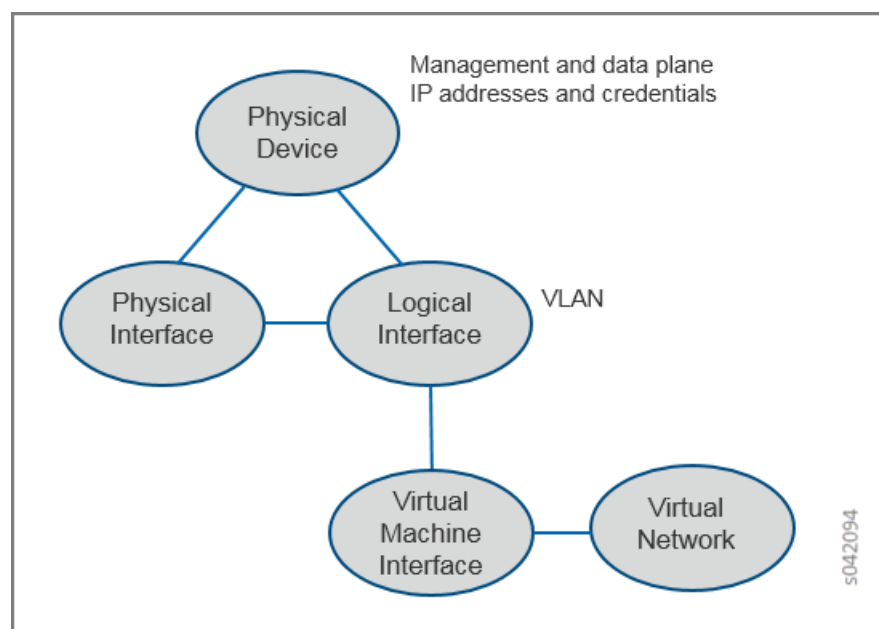
The Device Manager daemon listens to configuration events from the API server, creates any necessary configurations for all physical routers it is managing, and programs those physical routers.

You can extend a cluster to include physical Juniper Networks MX Series routers and other physical routers that support the Network Configuration (NETCONF) protocol. You can configure physical routers to be part of any of the virtual networks configured in the Contrail cluster, facilitating communication between the physical routers and the Contrail control nodes. Contrail policy configurations can be used to control this communication.

Configuration Model

[Figure 78 on page 245](#) depicts the configuration model used in the system. The Physical Router, Physical Interface, and Logical Interface all represent physical router entities.

Figure 78: Contrail Configuration Model



Configuring a Physical Router

The Contrail Web user interface can be used to configure a physical router into the Contrail system. Select **Configure > Physical Devices > Physical Routers** to create an entry for the physical router and provide the router's management IP address and user credentials.

The following shows how a Juniper Networks MX Series device can be configured from the Contrail Web user interface.

Figure 79: Add Physical Router Window

The screenshot shows a web form titled "Add Netconf Managed Physical Router". The form includes the following fields and controls:

- Name**: A text input field.
- Vendor**: A text input field.
- Model**: A text input field.
- Management IP**: A text input field.
- Netconf Username**: A text input field.
- Password**: A text input field.
- Junos Service Ports**: A checkbox that is checked.
- Port**: A text input field with a "+" icon to its right.
- SNMP Monitored**: A checkbox that is checked.

At the bottom right of the form are "Cancel" and "Save" buttons. A vertical text "s042493" is located on the right side of the form area.

Select **Configure > Physical Devices > Interfaces** to add the logical interfaces to be configured on the router. The name of the logical interface must match the name on the router (for example, ge-0/0/0.10).

Figure 80: Add Interface Window

Alternate Ways to Configure a Physical Router

You can also configure a physical router by using a Contrail REST API, see [“REST APIs for Extending the Contrail Cluster to Physical Routers, and Physical and Logical Interfaces”](#) on page 273.

Device Manager Configurations

Device Manager can configure all of the following on a Juniper Networks MX Series device and other physical routers.

- Create configurations for physical interfaces and logical interfaces as needed.
- Create VRF table entries as needed by the configuration.
- Add interfaces to VRF tables as needed.
- Create public VRF tables corresponding to external virtual networks.
- Create BGP protocol configuration for internal or external BGP groups as needed and adding iBGP and eBGP peers in appropriate groups.
- Program route-target import and export rules as needed by policy configurations.
- Create policies and firewalls as needed.
- Configure Ethernet VPNs (EVPNs).

Prerequisite Configuration Required on MX Series Device

Before using Device Manager to manage the configuration for an MX Series device, use the following Junos CLI commands to enable NETCONF on the device:

```
set system services netconf ssh
```

```
set system services netconf traceoptions file nc
```

```
set system services netconf traceoptions flag all
```

Debugging Device Manager Configuration

If there is any failure during a Device Manager configuration, the failed configuration is stored on the MX Series device as a candidate configuration. An appropriate error message is logged in the local system log by the Device Manager.

The log level in the Device Manager configuration file should be set to INFO for logging NETCONF XML messages sent to physical routers.

Configuration Scenarios

This section presents different configuration scenarios and shows snippets of generated MX Series configurations.

Configuring Physical Routers Using REST APIs

For information regarding configurations using REST APIs, see “REST APIs for Extending the Contrail Cluster to Physical Routers, and Physical and Logical Interfaces” on page 273.

Sample Python Script Using Rest API for Configuring an MX Device

Refer to the following link for a Python-based script for configuring required MX Series device resources in the Contrail system, using the VNC Rest API provided by Contrail.

https://github.com/Juniper/contrail-controller/blob/master/src/config/utils/provision_physical_router.py

Device Manager Functionality

Device Manager auto configures physical routers when it detects associations in the Contrail database.

The following naming conventions are used for generating MX Series router configurations:

- Device Manager generated configuration group name: `__contrail__`
- BGP groups:
 - Internal group name: `__contrail__`
 - External group name: `__contrail_external`
- VRF name: `_contrai_{l2|l3}_{vn-id}_{vn-name}`
- NAT VRF name: `_contrai_{l2|l3}_{vn-id}_{vn-name}-nat`

- Import policy: **[vrf-name]—import**, Export policy: **[vrf-name]—export**
- Service set: **sv-[vrf-name]**
- NAT rules, SNAT: **sv-[vrf-name]-sn-rule**, DNAT: **sv-[vrf-name]-dn-rule**
- SNAT term name: **term_[private_ip]**, DNAT term name: **term_[public_ip]**
- Firewall filters:
 - Public VRF filter: **redirect_to_public_vrf_filter**
 - Private VRF filter: **redirect_to_[vrf_name]_vrf**
- Logical interface unit numbers:
 - Service ports: **2*vn_id -1, 2*vn_id**
 - IRB interface: **vn_id**

Dynamic Tunnels

Dynamic tunnel configuration in Contrail allows you to configure GRE tunnels on the Contrail Web user interface. When Contrail detects this configuration, the Device Manager module constructs GRE tunnel configuration and pushes it to the MX Series router. A property named **ip-fabric-subnets** is used in the global system configuration of the Contrail schema. Each IP fabric subnet and BGP router is configured as a dynamic tunnel destination point in the MX Series router. The physical router data plane IP address is considered the source address for the dynamic tunnel. You must configure the data plane IP address for auto configuring dynamic tunnels on a physical router. The IP fabric subnets is a global configuration; all of the subnets are configured on all the physical routers in the cluster that have data plane IP configuration.

The following naming conventions are used in the API configuration:

- Global System Config: **ip-fabric-subnets**
- Physical Router: **data-plane-ip**

Web UI Configuration

[Figure 81 on page 250](#) shows the web user interface used to configure dynamic tunnels.

Figure 81: Edit Global Config Window

In the **Edit Global Config** window, the VTEP address is used for the **data-plane-ip** address.

The following is an example of the MX Series router configuration generated by the Device Manager.

```
root@host# show groups __contrail__ routing-options
router-id 172.16.184.200;
route-distinguisher-id 10.87.140.107;
autonomous-system 64512;
dynamic-tunnels {
  __contrail__ {
    source-address 172.16.184.200;
    gre;
    destination-networks {
      172.16.180.0/24;
      172.16.180.8/32;
    }
  }
}
```

```

172.16.185.200/32;
172.16.184.200/32;
172.16.180.5/32;
172.16.180.7/32;
}
}
}

```

BGP Groups

When Device Manager detects BGP router configuration and its association with a physical router, it configures BGP groups on the physical router.

Figure 82 on page 251 shows the web user interface used to configure BGP groups.

Figure 82: Edit BGP Router Window

The screenshot shows the 'Edit BGP Router' window with the following configuration:

- Hostname:** tasman
- Router Type:** ☐ Control Node, ☒ BGP Router
- Vendor ID:** mx
- IP Address:** 172.16.184.200
- Router ID:** 10.87.140.107
- Autonomous System:** 64512
- Address Families:** route-target x, inet-vpn, e-vpn x, inet6-vpn x
- Advanced Options:**
 - Peer(s) Selection:** (dropdown menu)
- Available Peer(s):** (list box)
- Configured Peer(s):** (list box)
- Buttons:** Cancel, Save

Figure 83 on page 252 shows the web user interface used to configure the physical router.

Figure 83: Edit Physical Router Window for BGP Groups

The screenshot shows the 'Edit Physical Router' configuration window. It includes the following fields:

- Name:** tasman
- Vendor:** juniper
- Model:** mx
- Management IP:** 10.87.140.107
- VTEP Address:** 172.16.184.200
- BGP Gateway:** tasman

A dropdown menu on the right side of the VTEP Address field shows the selected interface: s04/24/76.

The following is an example of the MX Series router configuration generated by the Device Manager.

```
root@host show groups __contrail__ protocols bgp
group __contrail__ {
    type internal;
    multihop;
    local-address 172.16.184.200;
    hold-time 90;
    keep all;
    family inet-vpn {
        unicast;
    }
    family inet6-vpn {
        unicast;
    }
    family evpn {
        signaling;
    }
    family route-target;
    neighbor 172.16.180.8;
    neighbor 172.16.185.200;
    neighbor 172.16.180.5;
    neighbor 172.16.180.7;
}

group __contrail_external__ {
    type external;
    multihop;
    local-address 172.16.184.200;
    hold-time 90;
    keep all;
    family inet-vpn {
        unicast;
    }
    family inet6-vpn {
        unicast;
    }
    family evpn {
        signaling;
    }
    family route-target;
}
```

Extending the Private Network

Device Manager allows you to extend a private network and ports to a physical router. When Device Manager detects a VNC configuration, it pushes Layer 2 (EVPN) and Layer 3 VRF, import and export rules and interface configuration to the physical router.

Figure 84 on page 253 shows the web user interface for configuring the physical router for extending the private network.

Figure 84: Edit Physical Router Window for Extending Private Networks

The screenshot shows the 'Edit Physical Router' window. It has a title bar 'Edit Physical Router'. Below it, there are several configuration fields:

- Name:** tasman
- Vendor:** juniper
- Model:** mx
- Management IP:** 10.87.140.107
- VTEP Address:** 172.16
- BGP Gateway:** tasman
- Virtual Networks:** A list containing two entries: 'vn_private-x1-63 (default-domain:default-project) x' and 'vn_private-x2-17 (default-domain:default-project) x'.

The following is an example of the MX Series router configuration generated by the Device Manager.

```
/* L2 VRF */

root@host# show groups __contrail__ routing-instances
_contrail_l2_147_vn_private-x1-63
vtep-source-interface lo0.0;
instance-type virtual-switch;
vrf-import _contrail_l2_147_vn_private-x1-63-import;
vrf-export _contrail_l2_147_vn_private-x1-63-export;
protocols {
    evpn {
        encapsulation vxlan;
        extended-vni-list all;
    }
}
bridge-domains {
    bd-147 {
        vlan-id none;
        routing-interface irb.147;
        vxlan {
            vni 147;
        }
    }
}
```

```
}

/* L3 VRF */
root@host# show groups __contrail__ routing-instances
_contrail_l3_147_vn_private-x1-63
instance-type vrf;
interface irb.147;
vrf-import _contrail_l3_147_vn_private-x1-63-import;
vrf-export _contrail_l3_147_vn_private-x1-63-export;
vrf-table-label;
routing-options {
    static {
        route 1.0.63.0/24 discard;
    }
    auto-export {
        family inet {
            unicast;
        }
    }
}

/* L2 Import policy */

root@host# ...cy-options policy-statement
_contrail_l2_147_vn_private-x1-63-import
term t1 {
    from community target_64512_8000066;
    then accept;
}
then reject;

/* L2 Export Policy */
root@host# ...ail__ policy-options policy-statement
_contrail_l2_147_vn_private-x1-63-export
term t1 {
    then {
        community add target_64512_8000066;
        accept;
    }
}

/* L3 Import Policy */

root@host# ...ail__ policy-options policy-statement
_contrail_l3_147_vn_private-x1-63-import
term t1 {
    from community target_64512_8000066;
    then accept;
}
then reject;

/*L3 Export Policy */
root@host# ...ail__ policy-options policy-statement
_contrail_l3_147_vn_private-x1-63-export
term t1 {
    then {
        community add target_64512_8000066;
        accept;
    }
}
```

```

    }
}

```

Extending the Public Network

When a public network is extended to a physical router, a static route is configured on the MX Series router. The configuration copies the next hop from the **public.inet.0** routing table to the **inet.0** default routing table, and copies a forwarding table filter from the **inet.0** routing table to the **public.inet.0** routing table. The filter is applied to all packets being looked up in the **inet.0** routing table and matches destinations that are in the subnet(s) for the public virtual network. The policy action is to perform the lookup in the **public.inet.0** routing table.

Figure 85 on page 255 shows the web user interface for extending the public network.

Figure 85: Edit Network Gateway Window

The following is an example of the MX Series router configuration generated by the Device Manager.

```

/* forwarding options */

root@host show groups __contrail__ forwarding-options
family inet {
  filter {
    input redirect_to_public_vrf_filter;
  }
}

```

```
/* firewall filter configuration */

root@host# show groups __contrail__ firewall family inet filter
redirect_to_public_vrf_filter

term term-_contrail_l3_184_vn_public-x1- {

    from {

        destination-address {

            20.1.0.0/16;

        }

    }

    then {

        routing-instance _contrail_l3_184_vn_public-x1-;

    }

}

term default-term {

    then accept;

}

/* L3 VRF static route 0.0.0.0/0 configuration */

root@host# ...instances _contrail_l3_184_vn_public-x1- routing-options static
route 0.0.0.0/0
next-table inet.0;
```

Ethernet VPN Configuration

For every private network, a Layer 2 Ethernet VPN (EVPN) instance is configured on the MX Series router. If any Layer 2 interfaces are associated with the virtual network, logical interfaces are also created under the bridge domain.

The following is an example of the MX Series router configuration generated by the Device Manager.

```
root@host# show groups __contrail__ routing-instances
_contrail_l2_147_vn_private-x1-63
vtep-source-interface lo0.0;
instance-type virtual-switch;
vrf-import _contrail_l2_147_vn_private-x1-63-import;
vrf-export _contrail_l2_147_vn_private-x1-63-export;
protocols {
    evpn {
        encapsulation vxlan;
        extended-vni-list all;
    }
}
bridge-domains {
    bd-147 {
```



```
        vlan-id none;

        interface ge-1/0/5.0;
        routing-interface irb.147;
        vxlan {
            vni 147;
        }
    }
}
```

Floating IP Addresses and Source Network Address Translation for Guest Virtual Machines and Bare Metal Servers

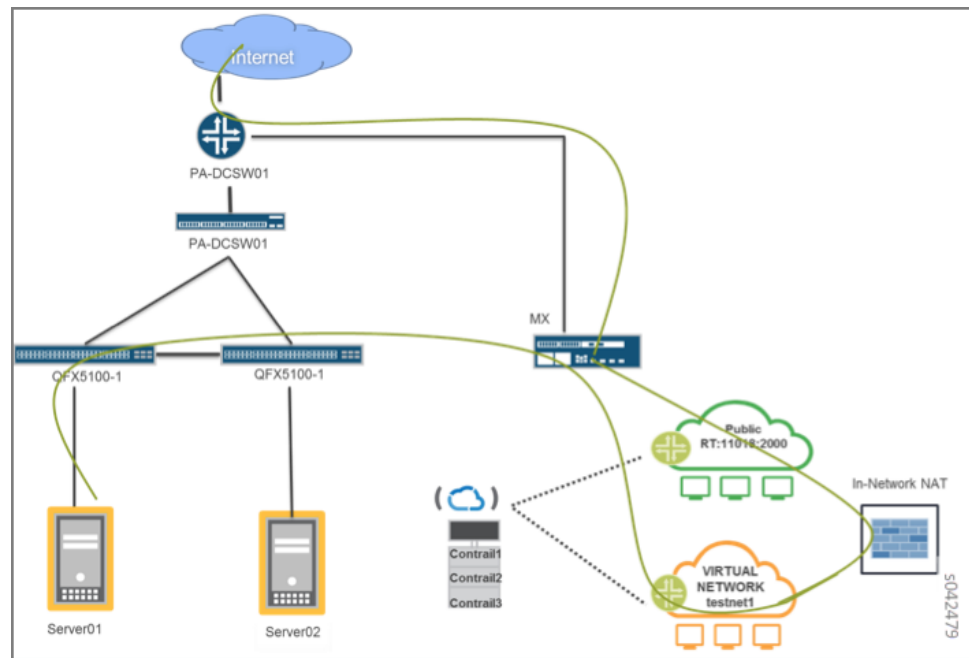
This section describes a bare metal server deployment scenario in which servers are connected to a TOR QFX device inside a private network and an MX Series router is the gateway for the public network connection.

The MX Series router provides the NAT capability that allows traffic from a public network to enter a private network and also allows traffic from the private network to the public network. To do this, you need to configure NAT rules on the MX Series router. The Device Manager is responsible for programming these NAT rules on MX Series routers when it detects that a bare metal server is connected to a public network.

You must configure virtual network computing for the TOR device, the MX Series router, the private network, and the public network, including the address pool. When a logical interface on the TOR device is associated with the virtual machine interface and a floating IP address is assigned to the same virtual machine interface (VMI), Contrail detects this and the Device Manager configures the necessary floating IP NAT rules on each of the MX Series routers associated with the private network.

[Figure 86 on page 258](#) illustrates that the Device Manager configures two special logical interfaces called *service-ports* on the MX Series router for NAT translation from the private network to the public network.

Figure 86: Logical Topology for Floating IP and SNAT



The Contrail schema allows a user to specify a service port name using the virtual network computing API. The service port must be a physical link on the MX Series router and the administrative and operational state must be up. The Device Manager creates two logical interfaces on this service port, one for each private virtual network, and applies NAT rules.

The private network routing instance on the MX Series router has a default static route (0.0.0.0/0) next hop pointing to the inside service interface. A public network routing instance on the MX Series router has a route for the private IP prefix next hop pointing to the outside service interface. The public IP address to private IP address and the reverse NAT rules are configured on the MX Series router.

A special routing instance for each private network to one or more public networks association is created on the MX Series router. This VRF has two interfaces on one side allowing traffic to and from the public network and another interface allowing traffic to and from the private network. Firewall filters on the MX Series router are configured so that, if the public network has floating IP addresses associated with a guest VM managed by the Contrail vRouter, the vRouter performs the floating IP address functionality. Otherwise, the MX Series router performs the NAT functions to send and receive the traffic to and from the bare metal server VM.

As illustrated in [Figure 86 on page 258](#), you must create the necessary physical device, interface, and virtual network configuration that is pushed to the MX Series router.

Contrail configuration can be done using the Web UI or VNC API. The required configuration is:

- Create the private virtual network.
- Create one or more TOR physical routers (No Junos OS configuration needs to be pushed to this device by Contrail. Therefore set the **vnc managed** attribute to **False**).
- Extend the private virtual network to the TOR device.
- Create physical and logical interfaces on the TOR device.
- Create the VMI on the private network for the bare metal server and associate the VMI with the logical interface. Doing that indicates that the bare metal server is connected to the TOR device through the logical interface. An instance IP address must be assigned to this VMI. The VMI uses a private IP address for the bare metal server.
- Create the gateway router. This is a physical router that is managed by the Device Manager.
- Configure the **service-port** physical interface information for the physical MX Series router. Device Manager configures two logical service interfaces on the MX Series router for each private network associated with the device, and automatically configures NAT rules on these interfaces for the private-to-public IP address translation and SNAT rules for the opposite direction. The logical port ID is calculated from the virtual network ID allocated by Contrail VNC. Two logical ports are required for each private network
- Associate the floating IP address, including creating the public network, the floating IP address pool, and a floating IP address in Contrail, and associate this IP address with the VMI bare metal server.
- The private network and public network must be extended to the physical router.

When the required configuration is present in Contrail, the Device Manager pushes the generated Junos OS configuration to the MX Series device. An example configuration is shown in the following.

```
/* NAT VRF configuration */

root@host# show groups __contrail__ routing-instances
_contrail_13_147_vn_private-x1-63-nat

instance-type vrf;

interface si-2/0/0.293;

vrf-import _contrail_13_147_vn_private-x1-63-nat-import;

vrf-export _contrail_13_147_vn_private-x1-63-nat-export;

vrf-table-label;

routing-options {
    static {
        route 0.0.0.0/0 next-hop si-2/0/0.293;
```

```
    }

    auto-export {

        family inet {

            unicast;

        }

    }

}

/* NAT VRF import policy */

root@host# ...y-statement _contrail_l3_147_vn_private-x1-63-nat-import

term t1 {

    from community target_64512_8000066;

    then accept;

}

then reject;

/* NAT VRF Export policy */

root@host# ..._ policy-options policy-statement
_contrail_l3_147_vn_private-x1-63-nat-export

term t1 {

    then reject;

}

/* The following additional config is generated for public l3 vrf */

root@host# show groups __contrail__ routing-instances
_contrail_l3_184_vn_public-x1-

interface si-2/0/0.294;

routing-options {

    static {

        route 20.1.252.8/32 next-hop si-2/0/0.294;

        route 20.1.252.9/32 next-hop si-2/0/0.294;

    }

}

/* Services set configuration */
```

```
root@host# show groups __contrail__

services {

  service-set sv-_contrail_l3_147_vn_ {

    nat-rules sv-_contrail_l3_147_vn_-sn-rule;

    nat-rules sv-_contrail_l3_147_vn_-dn-rule;

    next-hop-service {

      inside-service-interface si-2/0/0.293;

      outside-service-interface si-2/0/0.294;

    }

  }

}

/* Source Nat Rules*/

root@host# show groups __contrail__ services nat rule
sv-_contrail_l3_147_vn_-sn-rule

match-direction input;

term term_1_0_63_248 {

  from {

    source-address {

      1.0.63.248/32;

    }

  }

  then {

    translated {

      source-prefix 20.1.252.8/32;

      translation-type {

        basic-nat44;

      }

    }

  }

}

term term_1_0_63_249 {
```

```
    from {
        source-address {
            1.0.63.249/32;
        }
    }
    then {
        translated {
            source-prefix 20.1.252.9/32;
            translation-type {
                basic-nat44;
            }
        }
    }
}

/* Destination NAT rules */

root@host# show groups __contrail__ services nat rule
sv-__contrail__l3_147_vn_-dn-rule

match-direction output;

term term_20_1_252_8 {
    from {
        destination-address {
            20.1.252.8/32;
        }
    }
    then {
        translated {
            destination-prefix 1.0.63.248/32;
            translation-type {
                dnat-44;
            }
        }
    }
}
```

```
    }
  }
  term term_20_1_252_9 {
    from {
      destination-address {
        20.1.252.9/32;
      }
    }
    then {
      translated {
        destination-prefix 1.0.63.249/32;
        translation-type {
          dnat-44;
        }
      }
    }
  }
}

/* Public VRF Filter */
root@host# show groups __contrail__ firewall family inet filter
redirect_to_public_vrf_filter
term term-_contrail_l3_184_vn_public-x1- {
  from {
    destination-address {
      20.1.0.0/16;
    }
  }
  then {
    routing-instance _contrail_l3_184_vn_public-x1-;
  }
}
```

```
term default-term {
    then accept;
}

/* NAT Vrf filter */

root@host# ...all family inet filter
redirect_to__contrail_l3_147_vn_private-x1-63-nat_vrf

term term-_contrail_l3_147_vn_private-x1-63-nat {
    from {
        source-address {
            1.0.63.248/32;
            1.0.63.249/32;
        }
    }
    then {
        routing-instance _contrail_l3_147_vn_private-x1-63-nat;
    }
}

term default-term {
    then accept;
}

/* IRB interface for NAT VRF */

root@host# show groups __contrail__ interfaces

irb {
    gratuitous-arp-reply;
    unit 147 {
        family inet {
            filter {
                input redirect_to__contrail_l3_147_vn_private-x1-63-nat_vrf;
            }
            address 1.0.63.254/24;
        }
    }
}
```



```

    }

/* Service Interfaces config */

root@host# show groups __contrail__ interfaces si-2/0/0

unit 293 {

    family inet;

    service-domain inside;

}

unit 294 {

    family inet;

    service-domain outside;

}

```

Samples of Generated Configurations for an MX Series Device

This section provides several scenarios and samples of MX Series device configurations generated using Python script.

Scenario 1: Physical Router With No External Networks

The following describes the use case of basic **vn**, **vmi**, **li**, **pr**, **pi** configuration with no external virtual networks. When the Python script shown in the following is executed with the parameters of this use case, the configuration is applied on the MX Series physical router.

Script executed on the Contrail controller:

```
# python provision_physical_router.py --api_server_ip 127.0.0.1 --api_server_port
8082 --admin_user user1 --admin_password password1 --admin_tenant_name
default-domain --op add_basic
```

Generated configuration for MX Series device:

```

root@host# show groups __contrail__
routing-options {
    route-distinguisher-id 10.84.63.133;
    autonomous-system 64512;
}
protocols {
    bgp {
        group __contrail__ {
            type internal;
            multihop;
            local-address 10.84.63.133;
            keep all;
            family inet-vpn {
                unicast;
            }
            family inet6-vpn {
                unicast;
            }
        }
    }
}

```

```

    }
    family evpn {
        signaling;
    }
    family route-target;
}
group __contrail_external__ {
    type external;
    multihop;
    local-address 10.84.63.133;
    keep all;
    family inet-vpn {
        unicast;
    }
    family inet6-vpn {
        unicast;
    }
    family evpn {
        signaling;
    }
    family route-target;
}
}
}
policy-options {
    policy-statement __contrail__default-domain_default-project_vn1-export {
        term t1 {
            then {
                community add target_64200_8000008;
                accept;
            }
        }
    }
    policy-statement __contrail__default-domain_default-project_vn1-import {
        term t1 {
            from community target_64200_8000008;
            then accept;
        }
        then reject;
    }
    community target_64200_8000008 members target:64200:8000008;
}
routing-instances {
    __contrail__default-domain_default-project_vn1 {
        instance-type vrf;
        interface ge-1/0/5.0;
        vrf-import __contrail__default-domain_default-project_vn1-import;
        vrf-export __contrail__default-domain_default-project_vn1-export;
        vrf-table-label;
        routing-options {
            static {
                route 10.0.0.0/24 discard;
            }
            auto-export {
                family inet {
                    unicast;
                }
            }
        }
    }
}
}

```

Scenario 2: Physical Router With External Network, Public VRF

This section describes the use case of **vn**, **vmi**, **li**, **pr**, **pi** configuration with an external virtual network, public VRF. When the Python script shown is executed with the parameters of this use case, the configuration is applied on the MX Series physical router.

This example assumes that the configuration already described in Scenario 1 has been executed.

Script executed on the Contrail controller:

```
# python provision_physical_router.py --api_server_ip 127.0.0.1 --api_server_port
8082 --admin_user user1 --admin_password password1 --admin_tenant_name
default-domain --op add_basic --public_vrf_test True
```

Generated configuration for MX Series device:

The following additional configuration is pushed to the MX Series device, in addition to the configuration generated in Scenario 1.

```
forwarding-options {
  family inet {
    filter {
      input redirect_to__contrail__default-domain_default-project_vn1_vrf;
    }
  }
}
firewall {
  filter redirect_to__contrail__default-domain_default-project_vn1_vrf {
    term t1 {
      from {
        destination-address {
          10.0.0.0/24;
        }
      }
      then {
        routing-instance __contrail__default-domain_default-project_vn1;
      }
    }
    term t2 {
      then accept;
    }
  }
}
routing-instances {
  __contrail__default-domain_default-project_vn1 {
    routing-options {
      static {
        route 0.0.0.0/0 next-table inet.0;
      }
    }
  }
}
```

Scenario 3: Physical Router With External Network, Public VRF, and EVPN

The scenario in this section describes the use case of **vn, vmi, li, pr, pi** physical router configuration with external virtual networks (public VRF) and EVPN configuration. When the Python script (as in the previous examples) is executed with the parameters of this scenario, the following configuration is applied on the MX Series physical router.

This example assumes that the configuration already described in Scenario 1 has been executed.

Script executed on the Contrail controller:

```
# python provision_physical_router.py --api_server_ip 127.0.0.1 --api_server_port
8082 --admin_user user1 --admin_password password1 --admin_tenant_name
default-domain --op add_basic --public_vrf_test True -vxlan 2002
```

Generated configuration for MX Series device:

The following additional configuration is pushed to the MX Series device, in addition to the configuration generated in Scenario 1.

```
protocols {
  mpls {
    interface all;
  }
}
firewall {
  filter redirect_to___contrail___default-domain_default-project_vn1_vrf {
    term t1 {
      from {
        destination-address {
          10.0.0.0/24;
        }
      }
      then {
        routing-instance ___contrail___default-domain_default-project_vn1;
      }
    }
    term t2 {
      then accept;
    }
  }
}
routing-instances {
  ___contrail___default-domain_default-project_vn1 {
    vtep-source-interface lo0.0;
    instance-type virtual-switch;
    vrf-target target:64200:8000008;
    protocols {
      evpn {
        encapsulation vxlan;
        extended-vni-all;
      }
    }
    bridge-domains {
      bd-2002 {
```

```

        vlan-id 2002;
        interface ge-1/0/5.0;
        routing-interface irb.2002;
        vxlan {
            vni 2002;
            ingress-node-replication;
        }
    }
}

```

Scenario 4: Physical Router With External Network, Public VRF, and Floating IP Addresses for a Bare Metal Server

The scenario in this section describes the user case of **vn**, **vmi**, **li**, **pr**, **pi** physical router configuration with external virtual networks (public VRF) and floating IP addresses for bare metal server configuration.

Script executed on the Contrail controller:

```

#python provision_physical_router.py --api_server_ip <ip address> --api_server_port
8082 --admin_user admin --admin_password <password> --admin_tenant_name
default-domain --op {fip_test|delete_fip_test}

```

Related Documentation

- [REST APIs for Extending the Contrail Cluster to Physical Routers, and Physical and Logical Interfaces on page 273](#)

SR-IOV VF as the Physical Interface of vRouter

Starting with Contrail 3.0, support is provided for single-root I/O virtualization (SR-IOV) virtual functions used as the physical router for vRouter.

SR-IOV allows a network interface to separate the access to its resources across multiple PCI Express functions. The functions can be physical or virtual.

The Contrail vRouter can use an SR-IOV virtual function as its physical interface. One virtual function on a network interface can be used by vRouter, while the remaining virtual functions can be used by virtual machines on the same compute node. It is also possible to create a VLAN interface on a virtual function, and use that as the physical interface of the vRouter.

Alternatively, virtual functions from two different interfaces can be bonded together, and that bonded interface can be used as the physical interface of the vRouter. It is also possible to create a VLAN on a bonded interface, like the one just described, and then use that bonded interface as the physical interface of the vRouter.

To set up virtual functions for the physical interface of a vRouter:

1. Include the **env.sriov** section in the **testbed.py** file, and complete the following steps to define the SR-IOV virtual functions, so that the virtual functions are created during the provisioning of the cluster.
2. Within **env.sriov**, create SR-IOV virtual functions on the compute nodes (**host1** and **host2**, in this example). Virtual functions are usually identified with the following naming scheme: **p6p2_1**, **p6p2_2**, and so on. For example:

```
env.sriov = {  
  
    host1 : [ {'interface' : 'p6p2', 'VF' : 7, 'physnets' : ['physnet1']} ],  
    host2 : [ {'interface' : 'p6p2', 'VF' : 7, 'physnets' : ['physnet1']} ]  
  
}
```

3. Specify the virtual function interfaces in the **control_data** section of the **testbed.py** file, with or without a VLAN, so that they can be used by the vRouter. For example:

```
control_data = {  
  
    host1 : { 'ip': '10.x.x.100/2x', 'gw' : '10.x.x.254', 'device': 'p6p2_1' },  
    host2 : { 'ip': '10.x.x.200/2x', 'gw' : '10.x.x.254', 'device': 'p6p2_2' }  
  
}
```

4. Optionally, for bonded interfaces (**bond0** in this example), specify the virtual functions in the **bond** section of the **testbed.py** file, with or without a VLAN. For example:

```
bond= {  
  
    host1 : { 'name': 'bond0', 'member': ['p6p2_4', 'p6p1_5'], 'mode': 'balance-xor' },  
    host2 : { 'name': 'bond0', 'member': ['p6p2_2', 'p6p1_3'], 'mode': 'balance-xor' }  
  
}
```

Using Gateway Mode to Support Remote Instances

Extending virtual instances running non-Openstack clusters or extending bare metal servers into Contrail virtual networks can be achieved using the OVSDB protocol.

Additionally, starting with Contrail Release 3.1, an experimental mode has been added that enables you to configure a Contrail compute node to run in gateway mode to support remote instances.

With Contrail Release 3.2, the gateway mode can also be used with VMware, using the VMware VMs as the remote instances.

- [Gateway Mode Overview on page 271](#)
- [Provisioning Gateway Mode on page 271](#)
- [Configuring Gateway Mode on page 271](#)
- [Configuring Gateway Mode and High Availability on page 272](#)
- [Scaling on page 272](#)

Gateway Mode Overview

Traffic from each external virtual instance is tagged with a unique VLAN, which is then mapped to a virtual machine interface (VMI) in the Contrail cluster. A Contrail compute node can be configured to map VLAN-tagged traffic coming on a physical port, other than the cluster's underlay IP fabric port, to a VMI configured in the Contrail cluster. The VMI corresponds to the interface of the remote instance. A vRouter on a gateway compute node operates like a local VMI, with traffic subjected to the forwarding decisions and policies that would be on the local VMI.



NOTE: For gateway mode, one VLAN is mapped to one virtual machine interface.

Provisioning Gateway Mode

To provision gateway mode:

In `/etc/contrail/contrail-vrouter-agent.conf`, in the **DEFAULT** section add:

```
gateway_mode = server
```

When finished, restart **contrail-vrouter-agent**.

Configuring Gateway Mode

To configure gateway mode:

1. Configure a physical router, using the host name of the compute node that acts as the gateway.
2. Create a physical interface on the physical router, using the name of the interface on the compute node that will be used for the gateway traffic.
3. Create a logical interface on the physical interface, using a unique VLAN ID and set the type to L2.
4. Create a virtual machine interface (VMI) in the required virtual network, identifying the MAC address and IP address of the remote instance, then link the VMI to the logical interface.

External Configuration

The traffic from the remote instance should come to the gateway port with the required VLAN tag.

Configuring Gateway Mode and High Availability

Multiple gateway nodes can be configured to have high availability.

The selection of the active gateway node is expected to be handled by using (R)STP from the switches connecting the gateway node. For this, a special virtual network is configured in Contrail that will flood the STP BPDUs.

For high availability for the gateway mode, make the following changes to the gateway mode configuration procedure:

1. On the gateway compute nodes, create logical interfaces with VLAN 0.
2. Create a dummy VMI belonging to the special virtual network, following the regular gateway mode configuration procedure.
3. Link the VMI to the VLAN 0 logical interfaces on the gateway nodes that will form a high availability group. This enables STP to allow traffic to one of the gateway ports, while blocking others.
4. For each remote instance, create a logical interface on the gateway nodes in the high availability group. Link the logical interface to the VMI created for the remote instance. The corresponding instance IP should have active-backup mode set, which is the default mode.

Upon completion of this procedure, Contrail will handle the switch over of the traffic to a different gateway node.



NOTE: This procedure can also be used to set up a vCenter gateway. Because the VMware VMs are the remote instances, their traffic must be configured to arrive VLAN-tagged at the gateway node's physical interface and their interfaces must be configured in Contrail. Using this configuration, all Contrail features available on a virtual machine interface will be applied for any traffic between VMware and Contrail.

Scaling

The default number of interfaces supported on a compute node is 4352. Because each remote instance has a logical interface and a virtual machine interface, up to 2K remote interfaces can be supported with the default configuration. To support 4K remote instances, the maximum number of interfaces on the compute node that is acting as the gateway should be configured to be 8K. For more information about how the vRouter

options can be modified, see
<https://github.com/Juniper/contrail-controller/wiki/Vrouter-Module-Parameters>.

REST APIs for Extending the Contrail Cluster to Physical Routers, and Physical and Logical Interfaces

Introduction: REST APIs for Extending Contrail Cluster

Use the following REST APIs when extending the Contrail cluster to include physical routers, physical interfaces, and logical interfaces.

REST API for Physical Routers

Use the following REST API when extending the Contrail cluster to include physical routers.

```
{
  u'physical-router': {
    u'physical_router_management_ip': u'100.100.100.100',
    u'virtual_router_refs': [],
    u'fq_name': [
      u'default-global-system-config',
      u'test-router'
    ],
    u'name': u'test-router',
    u'physical_router_vendor_name': u'juniper',
    u'parent_type': u'global-system-config',
    u'virtual_network_refs': [],
    'id_perms': {
      u'enable': True,
      u'uuid': None,
      u'creator': None,
      u'created': 0,
      u'user_visible': True,
      u'last_modified': 0,
      u'permissions': {
        u'owner': u'cloud-admin',
        u'owner_access': 7,

```

```
        u'other_access': 7,
        u'group': u'cloud-admin-group',
        u'group_access': 7
    },
    u'description': None
},
u'bgp_router_refs': [],
u'physical_router_user_credentials': {
    u'username': u'',
    u'password': u''
},
'display_name': u'test-router',
u'physical_router_dataplane_ip': u'101.1.1.1'
}
```

REST API for Physical Interfaces

Use the following REST API when extending the Contrail cluster to include physical interfaces.

```
{
  u'physical-interface': {
    u'parent_type': u'physical-router',
    'id_perms': {
      u'enable': True,
      u'uuid': None,
      u'creator': None,
      u'created': 0,
      u'user_visible': True,
      u'last_modified': 0,
      u'permissions': {
        u'owner': u'cloud-admin',
```

```
        u'owner_access': 7,
        u'other_access': 7,
        u'group': u'cloud-admin-group',
        u'group_access': 7
    },
    u'description': None
},
u'fq_name': [
    u'default-global-system-config',
    u'test-router',
    u'ge-0/0/1'
],
u'name': u'ge-0/0/1',
'display_name': u'ge-0/0/1'
}
}
```

REST API for Logical Interfaces

Use the following REST API when extending the Contrail cluster to include logical interfaces.

```
{
  u'logical-interface': {
    u'fq_name': [
      u'default-global-system-config',
      u'test-router',
      u'ge-0/0/1',
      u'ge-0/0/1.0'
    ],
    u'parent_uuid': u'6608b8ef-9704-489d-8cbc-fed4fb5677ca',
    u'logical_interface_vlan_tag': 0,
    u'parent_type': u'physical-interface',
    u'virtual_machine_interface_refs': [
```

```
    {
      u'to': [
        u'default-domain',
        u'demo',
        u'4a2edbb8-b69e-48ce-96e3-7226c57e5241'
      ]
    },
    'id_perms': {
      u'enable': True,
      u'uuid': None,
      u'creator': None,
      u'created': 0,
      u'user_visible': True,
      u'last_modified': 0,
      u'permissions': {
        u'owner': u'cloud-admin',
        u'owner_access': 7,
        u'other_access': 7,
        u'group': u'cloud-admin-group',
        u'group_access': 7
      },
      u'description': None
    },
    u'logical_interface_type': u'l2',
    'display_name': u'ge-0/0/1.0',
    u'name': u'ge-0/0/1.0'
  }
}
```

- Related Documentation**
- [Using ToR Switches and OVSDDB to Extend the Contrail Cluster to Other Instances on page 229](#)
 - [Using Device Manager to Manage Physical Routers on page 244](#)

CHAPTER 10

Installing and Using Contrail Storage

- [Installing and Using Contrail Storage on page 279](#)

Installing and Using Contrail Storage

- [Overview of the Contrail Storage Solution on page 279](#)
- [Basic Storage Functionality with Contrail on page 280](#)
- [Ceph Block and Object Storage Functionality on page 280](#)
- [Using the Contrail Storage User Interface on page 280](#)
- [Hardware Specifications on page 282](#)
- [Contrail Storage Provisioning on page 282](#)

Overview of the Contrail Storage Solution

Contrail provides a storage support solution using OpenStack Cinder configured to work with Ceph. Ceph is a unified, distributed storage system whose infrastructure provides storage services to Contrail.

The Contrail storage solution has the following features:

- Provides storage class features to Contrail clusters, including replication, reliability, and robustness.
- Uses open source components.
- Uses Ceph block and object storage functionality.
- Integrates with OpenStack Cinder functionality.
- Does not require virtual machines (VMs) to configure mirrors for replication.
- Allows nodes to provide both compute and storage services.
- Provides easy installation of basic storage functionality based on Contrail roles.
- Provides a Contrail-integrated user interface from which the user can monitor Ceph components and drill down for more information about components.
- Provides native live-migration support if the VM is booted with Ceph storage as its root volume.
- Provides object storage support through Swift and S3 APIs.

Basic Storage Functionality with Contrail

The following are basic interaction points between Contrail and the storage solution.

- Cinder volumes must be manually configured prior to installing the Contrail storage solution. The Cinder volumes can be attached to virtual machines (VMs) to provide additional storage.
- The storage solution stores virtual machine boot images and snapshots in Glance, using Ceph object storage functionality.
- All storage nodes can be monitored through a graphical user interface (GUI).
- It is possible to migrate virtual machines that have ephemeral storage in Ceph.

Ceph Block and Object Storage Functionality

In Contrail Release 4.0, installing the Contrail storage solution creates the following Ceph configurations.

- Each disk is configured as a standalone storage device, enhancing optimal performance and creating proper failure boundaries. Ceph allocates and assigns a process called object storage daemon (OSD) to each disk.
- A replication factor of 2 is configured, consisting of one original instance plus one replica copy. Ceph ensures that each replica is on a different storage node.
- A Ceph monitor process (mon) is configured on the contrail-ceph-controller node.
- The correct number of placement groups are automatically configured, based on the number of disk drives in the cluster.
- Properly identified SSD drives are set up for use as Ceph OSD journals to reduce write latencies.
- Multi-pool configuration is set up to segregate the OSD disks into logical pools improving performance and efficiency.
- If multiple storage nodes are in a single chassis, the **chassis** option helps in defining replication of data and also disabling replication of data within the nodes of the same chassis. Replication helps in avoiding data loss during a power failure to the chassis.

Using the Contrail Storage User Interface

The Contrail storage solution provides a user interface integrated into the Contrail user interface. The storage solution user interface displays the following:

- Customer usable space, which is different from Ceph total space. The displayed usable space does not display the space used by replication and other Ceph functions.
- Monitor OSDs (disks), monitoring processes (mon), and state changes, enabling quick identification of resource failures within storage components.
- Total cluster I/O statistics and individual drive statistics.

- Ceph-specific information about each OSD (disk).
- Ceph logs, Ceph nodes, and Ceph alerts.

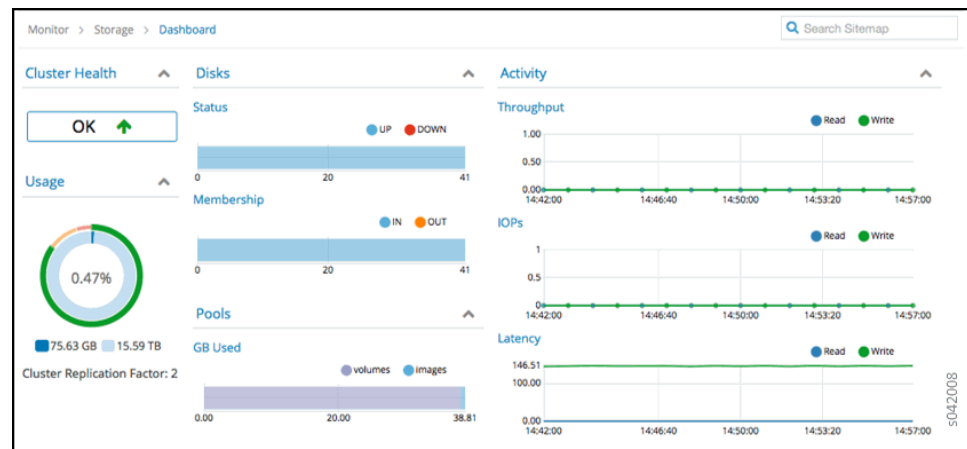
Select **Monitor > Infrastructure > Dashboard** to display an at-a-glance view of the system infrastructure components, including the numbers of virtual routers, control nodes, analytics nodes, config nodes, and storage nodes currently operational, and a bubble chart of storage nodes showing the Available (%) and Total Storage (GB). See the following figure.



Bubble charts use the following color-coding scheme for storage nodes:

- Blue—working as configured.
- Red—error, node is down.
- Yellow—one of the node disks is down.

Select **Monitor > Storage > Dashboard** to see a summary of cluster health, usage, pools, and disk status, and to gain insight into activity statistics for all nodes. See the following figure.



Hardware Specifications

The following are additional hardware specifications needed for the Contrail storage solution.

Additional minimum specifications:

- Two 500 GB, 7200 RPM drives in the server 4 and server 5 cluster positions (those with the compute storage role) in the Contrail installation. This configuration provides 1 TB of clustered, replicated storage.

Recommended compute storage configuration:

- For every 4-5 HDD devices on one compute storage node, use one SSD device to provide the OSD journals for that set of HDD devices.

Contrail Storage Provisioning

The **contrail-ceph-controller** and **contrail-ceph-compute** are two roles required to enable Ceph storage. The **contrail-ceph-controller** role is added to the Ceph monitor servers. The number of mons is limited to three for small clusters and five for large clusters with more than 1000 disks. The **contrail-ceph-compute** role is added to the servers that have the physical disks required for Ceph storage and also to the OpenStack Nova compute nodes that require Ceph storage services.

The following example displays sample **cluster.json** to provide Ceph storage configurations.

```
"parameters": {
  "provision": {
    "contrail_4": {
      "storage_ceph_config": {
        "replica_size": 2,
        "ceph_object_storage": "True",
        "object_store_pool": "volumes"
      }
    }
  }
}
```

The **replica_size** is added to change the default replica size of 2. The **ceph_object_storage** option enables the Ceph-based object storage to support Swift and S3 APIs and the **object_store_pool** option specifies the Ceph pool used for the Ceph object storage functionality.

The following example displays sample **server.json** to enable Ceph storage.

Server.json :

```
"parameters": {
  "provision": {
    "contrail_4": {
      "storage": {
        "storage_osd_disks": [
          "/dev/sdb:/dev/sdd:Pool_1",

```

```

        "/dev/sdc:/dev/sdd:Pool_2"
    ],
    "storage_osd_ssd_disks": [
        "/dev/sde:Pool_1",
        "/dev/sdf:Pool_2"
    ],
    "chassis_id": "chassis_1"
  }
}
"roles": [
  "contrail-ceph-controller", "contrail-ceph-compute"
  [p0-
    ]
]

```

The **storage_osd_disks** or **storage_osd_ssd_disk** is needed to provision the disks for Ceph. The first disk is OSD disk and the second optional disk is used as a Journal disk. If a multi-pool configuration is required, the pool name can be added along the OSD disk as shown in the **server.json** to enable Ceph storage. The **chassis_id** option can also be included per server. Pools and the **chassis** option cannot co-exist.



NOTE: The disks added to Ceph are not included in the OS disk. The **partition** parameter in the server JSON lists only the required OS disks.

```

"parameters": {
  "partition": "/dev/sda"
}

```

The disks added to Ceph cannot be part of LVM.

Upgrading Contrail Software

- [Upgrading Contrail 3.2 to 4.0 on page 285](#)
- [Dynamic Kernel Module Support \(DKMS\) for vRouter on page 288](#)

Upgrading Contrail 3.2 to 4.0

Contrail Release 4.0 presents a number of differences from earlier versions of Contrail, the most notable is that many Contrail services are now run within containers.

This section provides the process for upgrading an existing Contrail Release 3.2 system to Contrail Release 4.0. You can perform an upgrade by using a convenient script, or by performing the upgrade manually. Both procedures are included in this topic.

- [Overview of Changes in Contrail Release 4.0 on page 285](#)
- [Using a Server Manager Script to Upgrade from Contrail 3.2 to 4.0 on page 286](#)
- [Steps for Upgrading Contrail 3.2 to 4.0 \(Without using Server-Manager for upgrade\) on page 286](#)

Overview of Changes in Contrail Release 4.0

For previous releases of Contrail, through Contrail Release 3.2.x, Contrail could be provisioned by using fab commands or by using the Contrail Server Manager. Starting with Contrail Release 4.0, many Contrail services have been containerized, and provisioning can only be accomplished by using the Contrail Server Manager.

Additionally, the use of fab commands is no longer supported in Contrail Release 4.0.

You can perform the upgrade by using the steps presented here, or by using a script available in Server Manager that automates the steps.

Significant differences between Contrail 3.2 and 4.0 include:

- A number of Contrail services are run in containers. Container default names in Contrail Release 4.0 include:
 - contrail-controller
 - contrail-analytics

- contrail-analyticsdb
- contrail-lb (load-balancer)

For more information about Contrail containers, see [“Introduction to Containerized Contrail Modules” on page 11](#).

Assumptions

The upgrade procedure assumes that RabbitMQ server and Neutron server are running on the OpenStack node by default.

Using a Server Manager Script to Upgrade from Contrail 3.2 to 4.0

A script is available in Server Manager that runs the steps provided in *Steps for Upgrading Contrail 3.2 to 4.0*.

The script location is: `/opt/contrail/server_manager/inplace_upgrade.py`.

Information regarding using the script:

- The script uses the **openstack-config** utility, you can install the contrail-setup package to get the utility.
- The timeout for cluster provisioning completion is 3600 seconds, you can increase the timeout for a cluster with a large number of nodes.

The default timeout parameter in the script:

PROV_TIMEOUT = 3600

- The script execution log is saved at `/var/log/contrail/inplace_upgrade.log`.

Running the Upgrade Script

1. Add the image, cluster, and servers to Server Manager running Contrail Release 4.0 code.
2. Run the script.

```
user@node:/opt/contrail/server_manager# python inplace_upgrade.py -h
usage: inplace_upgrade.py [-h] cluster_name image_name

positional arguments:
  cluster_name  cluster to be upgraded.The cluster and servers should be
                existing in SM
  image_name    version to upgrade to

optional arguments:
  -h, --help    show this help message and exit
```

Steps for Upgrading Contrail 3.2 to 4.0 (Without using Server-Manager for upgrade)

This procedure lists the steps needed for upgrading Contrail 3.2 to 4.0.

This procedure is an in-service upgrade from a running 3.2 system that will temporarily run in parallel with the new 4.0 system.

The services and roles referred to in this procedure could be running in separate nodes. Run the procedure commands in the nodes that are running the relevant roles or services.

1. Stop the listed services.

- supervisor-analytics
- supervisor-support-service
- supervisor-database
- contrail-database
- supervisor-webui
- supervisor-config
- supervisor-control
- haproxy (all nodes)
- redis-server (analytics/webui node)
- memcached (config nodes)
- neutron-server (config nodes)
- zookeeper (config nodes)

Example stop commands include:

```
service supervisor-analytics stop
```

```
service supervisor-support-service stop
```

2. Stop the epmd process. The epmd process is a RabbitMQ service that is usually running on a config node.

3. Use a **status** command to verify that none of the services that were stopped in Step 1 are still running.

Example status commands include:

```
service supervisor-analytics status
```

```
service supervisor-support-service status
```

4. Prepare the cluster.json and server.json, as required for Server Manager provisioning for Contrail Release 4.0, see [“Installing Containerized Contrail Clusters Using Server Manager” on page 17](#).

5. Provision the new 4.0 cluster and wait for provisioning to be completed.

6. Check the status of provisioning.

```
server-manager status server --cluster_id <cluster-id>
```

7. To prevent port conflict with Cassandra between the two versions of Contrail, on the 3.2 config node(s), edit the Cassandra configuration file at `/etc/cassandra/cassandra.yaml` to change the listening port from 9160 to 29160.

When finished, start the Cassandra service.

```
rpc_port: 29160
```

```
service cassandra start
```

8. Identify the new (4.0) and old (3.2) IP address lists for Cassandra.
 - a. Connect to the controller container.
 - b. Edit `/usr/lib/python2.7/dist-packages/contrail_issu/issu_contrail_config.py` and add the old Cassandra list and the new Cassandra list:

```
'old_cassandra_address_list': '192.xxx.xxx.102:29160',
```

```
'new_cassandra_address_list': '10.xx.5.xxx:9161',
```

It is sufficient to provide a single ip:port of the Contrail 3.2 Cassandra node.

9. Run a Cassandra sync between the old and new config nodes, to copy the cassandra config keyspaces from the release 3.2 nodes to the Contrail 4.0 Cassandra database.

```
contrail-issu-pre-sync
```

Wait for the command to complete.

10. Shut down the Cassandra service in the 3.2 controller.
11. The upgrade of the compute nodes includes a reboot of the compute nodes. The reboot turns off the guest virtual machine instances. To restart the virtual machine instances, log in to the Openstack node and restart the VMs:

```
nova start <instance-uuid>
```

Dynamic Kernel Module Support (DKMS) for vRouter

Dynamic Kernel Module Support (DKMS) is a framework provided by Linux to automatically build out-of-tree driver modules for Linux kernels whenever the Linux distribution upgrades the existing kernel to a newer version.

In Contrail, the vRouter kernel module is an out-of-tree, high performance packet forwarding module that provides advanced packet forwarding functionality in a reliable and stable manner. Contrail provides a DKMS-compatible source package for Ubuntu so that if you deploy an Ubuntu-based Contrail system you do not need to manually compile the kernel module each time the Linux deployment gets upgraded.

The **contrail-vrouter-dkms** package provides the DKMS compatibility for Contrail. Prior to installing the **contrail-vrouter-dkms** package, you must install both the DKMS package

and the **contrail-vrouter-utils** package, because the **contrail-vrouter-dkms** package is dependent on both. Installing the **contrail-vrouter-dkms** package adds the vRouter sources to the DKMS database, builds the vRouter module, and installs it in the existing kernel modules tree. When a kernel upgrade occurs, DKMS ensures that the module is compiled for the newer kernel and installed in the proper location so that upon reboot, the newer module can be used with the upgraded kernel.

For more information about DKMS, refer to:

- DKMS Ubuntu documentation at <https://help.ubuntu.com/community/DKMS>
- DKMS Ubuntu manual pages at <http://manpages.ubuntu.com/manpages/lucid/man8/dkms.8.html>
- Linux Journal article on DKMS at <http://www.linuxjournal.com/article/6896>

PART 3

Configuring Contrail

- [Configuring Virtual Networks on page 293](#)
- [Example of Deploying a Multi-Tier Web Application Using Contrail on page 321](#)
- [Configuring Services on page 333](#)
- [Configuring Service Chaining on page 357](#)
- [Examples: Configuring Service Chaining on page 397](#)
- [Adding Physical Network Functions in Service Chains on page 413](#)
- [Configuring High Availability on page 423](#)
- [Multitenancy Support on page 435](#)
- [Load Balancers on page 453](#)
- [Optimizing Contrail on page 473](#)

CHAPTER 12

Configuring Virtual Networks

- [Creating Projects in OpenStack for Configuring Tenants in Contrail on page 294](#)
- [Creating a Virtual Network with Juniper Networks Contrail on page 296](#)
- [Creating a Virtual Network with OpenStack Contrail on page 299](#)
- [Creating an Image for a Project in OpenStack Contrail on page 301](#)
- [Creating a Floating IP Address Pool on page 304](#)
- [Using Security Groups with Virtual Machines \(Instances\) on page 305](#)
- [Support for IPv6 Networks in Contrail on page 308](#)
- [Configuring EVPN and VXLAN on page 311](#)

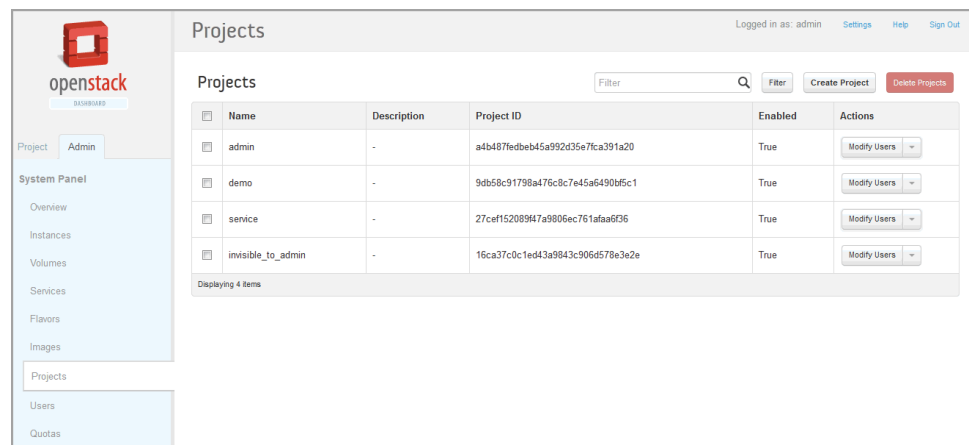
Creating Projects in OpenStack for Configuring Tenants in Contrail

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

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

1. Click the **Admin** tab on the OpenStack dashboard, then click the **Projects** link to access the **Projects** page; see [Figure 87 on page 294](#).

Figure 87: OpenStack Projects



2. In the upper right, click the **Create Project** button to access the **Add Project** window; see [Figure 88 on page 294](#).

Figure 88: Add Project

Add Project

Project Info | Project Members | Quota

Name
customer 1

Description
Additional information here...

Enabled
☒

From here you can create a new project to organize users.

Cancel Finish

3. In the **Add Project** window, on the **Project Info** tab, enter a **Name** and a **Description** for the new project, and select the **Enabled** check box to activate this project.
4. In the **Add Project** window, select the **Project Members** tab, and assign users to this project. Designate each user as **admin** or as **Member**.

As a general rule, one person should be a super user in the **admin** role for all projects and a user with a **Member** role should be used for general configuration purposes.
5. Click **Finish** to create the project.

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

- Related Documentation**
- [Creating a Virtual Network with Juniper Networks Contrail on page 296](#)
 - [Creating a Virtual Network with OpenStack Contrail on page 299](#)
 - [OpenStack documentation](#)

Creating a Virtual Network with Juniper Networks Contrail

Contrail makes creating a virtual network very easy for a self-service user. You create networks and network policies at the user dashboard, then associate policies with each network. The following procedure shows how to create a virtual network when using Juniper Networks Contrail.

1. You need to create an IP address management (IPAM) for your project for to create a virtual network. Select **Configure > Networking > IP Address Management**, then click the **Create** button.

The **Add IP Address Management** window appears, see [Figure 89 on page 296](#).

Figure 89: Add IP Address Management

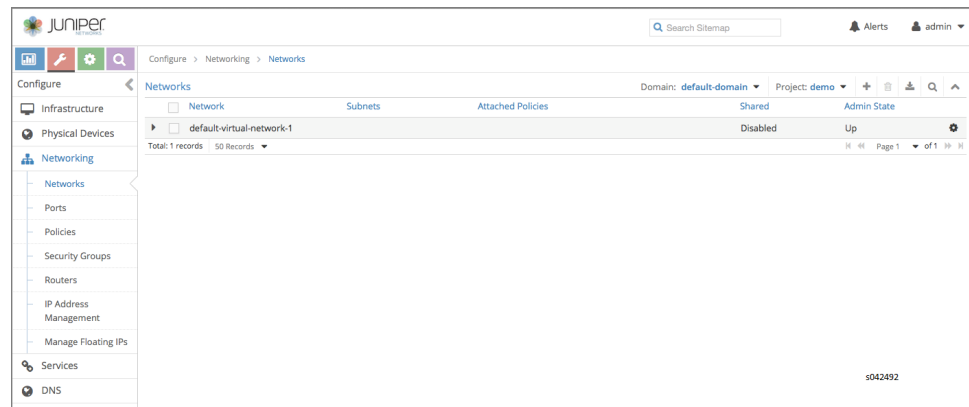
2. Complete the fields in **Add IP Address Management**: The fields are described in [Table 29 on page 296](#).

Table 29: Add IP Address Management Fields

Field	Description
Name	Enter a name for the IPAM you are creating.
DNS Method	Select from a list the domain name server method for this IPAM: Default , Virtual DNS , Tenant , or None .
NTP Server IP	Enter the IP address of an NTP server to be used for this IPAM.
Domain Name	Enter a domain name to be used for this IPAM.

3. Select **Configure > Networking > Networks** to access the **Configure Networks** page; see [Figure 90 on page 297](#).

Figure 90: Configure Networks




- Verify that your project is displayed as active in the upper-right field, then click the  icon. The **Create Network** window is displayed. See [Figure 91 on page 297](#). Use the scroll bar to access all sections of this window.

Figure 91: Create Network

- Complete the fields in the **Create Network** window with values that identify the network name, network policy, and IP options as needed. See field descriptions in [Table 30 on page 297](#).

Table 30: Create Network Fields

Field	Description
Name	Enter a name for the virtual network you are creating.
Network Policy	Select the policy to be applied to this network from the list of available policies. You can select more than one policy by clicking each one needed.

Table 30: Create Network Fields (*continued*)

Field	Description
Subnets	Use this area to identify and manage subnets for this virtual network. Click the + icon to open fields for IPAM, CIDR, Allocation Pools, Gateway, DNS, and DHCP. Select the subnet to be added from a drop down list in the IPAM field. Complete the remaining fields as necessary. You can add multiple subnets to a network. When finished, click the + icon to add the selections into the columns below the fields. Alternatively, click the - icon to remove the selections.
Host Routes	Use this area to add or remove host routes for this network. Click the + icon to open fields where you can enter the Route Prefix and the Next Hop. Click the + icon to add the information, or click the - icon to remove the information.
Advanced Options	Use this area to add or remove advanced options, including identifying the Admin State as Up or Down, to identify the network as Shared or External, to add DNS servers, or to define a VxLAN Identifier.
Floating IP Pools	Use this area to identify and manage the floating IP address pools for this virtual network. Click the + icon to open fields where you can enter the Pool Name and Projects. Click the + icon to add the information, or click the - icon to remove the information.
Route Target	Move the scroll bar down to access this area, then specify one or more route targets for this virtual network. Click the + icon to open fields where you can enter route target identifiers. Click the + icon to add the information, or click the - icon to remove the information.

- To save your network, click the **Save** button, or click **Cancel** to discard your work and start over.

Now you can create a network policy, see *Creating a Network Policy—Juniper Networks Contrail*.

Related Documentation

- [Creating an Image for a Project in OpenStack Contrail on page 301](#)
- *Launching a Virtual Machine (Instance)*
- *Creating a Network Policy—Juniper Networks Contrail*
- *Deleting a Virtual Network—Juniper Networks Contrail*

Creating a Virtual Network with OpenStack Contrail

Contrail makes creating a virtual network very easy for you. You create networks and network policies at the user dashboard, then associate policies with each network. The following procedure shows how to create a virtual network when using OpenStack.

1. To create a virtual network when using OpenStack Contrail, select **Project > Other > Networking**. The **Networks** window is displayed. See [Figure 92 on page 299](#).

Figure 92: Networks Window

Network	Summary	Status	Actions
default-virtual-network	0 instances in 0 IP Blocks, 0 available IP addresses	Up	Edit IP Blocks

Displaying 1 item

2. Verify that the correct project is displayed in the **Current Project** box, then click **Create Network**. The **Create Network** window is displayed. See [Figure 93 on page 299](#) and [Figure 94 on page 300](#).

Figure 93: Create Network Window

Create Network

Network * Subnet * Subnet Detail Associate Network Policies >

Network Name *

☒ Admin State

From here you can create a new network. In addition a subnet associated with the network can be created in the next panel and Policies can be associated to this network

Cancel Create

Figure 94: Create Network Window Subnet Tab

Create Network

Network • Subnet • Subnet Detail Associate Network Policies

☒ Create Subnet

Subnet Name

IPAM

default-network-ipam (default-project)

Network Address

IP Version *

IPv4

Gateway IP

☐ Disable Gateway

You can create a subnet associated with the new network, in which case "Network Address" must be specified. If you wish to create a network WITHOUT a subnet, uncheck the "Create Subnet" checkbox.

Cancel Create

s018524

- Click the **Network**, **Subnet**, **Subnet Detail**, and **Associate Network Policies** tabs to complete the fields in the **Create Network** window. See field descriptions in [Table 31 on page 300](#).

Table 31: Create Network Fields

Field	Description
Network Name	Enter a name for the network.
Subnet Name	Enter a name for the subnetwork.
IPAM	Select the IPAM associated with the IP block. For new projects, an IPAM can be added while creating the virtual network. VM instances created in this virtual network are assigned an address from this address block automatically by the system when a VM is launched.
Network Address	Enter the network address in CIDR format.
IP Version*	Select IPv4 or IPv6.

Table 31: Create Network Fields (*continued*)

Field	Description
Gateway IP	Optionally, enter an explicit gateway IP address for the IP address block. Check the Disable Gateway box if no gateway is to be used.
Network Policy	Any policies already created are listed. To select a policy, click the check box for the policy.

- Click the **Subnet Details** tab to specify the Allocation Pool, DNS Name Servers, and Host Routes.
- Click the **Associate Network Policies** tab to associate policies to the network.
- To save your network, click **Create Network**, or click **Cancel** to discard your work and start over.

Related Documentation

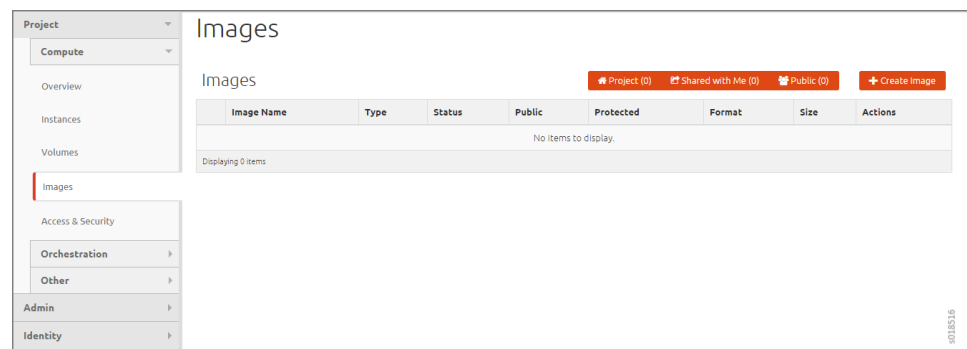
- *Deleting a Virtual Network—OpenStack Contrail*

Creating an Image for a Project in OpenStack Contrail

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

- In OpenStack, select **Project > Compute > Images**. The Images window is displayed. See [Figure 95 on page 301](#).

Figure 95: OpenStack Images Window



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

The **Create An Image** window is displayed. See [Figure 96 on page 302](#).

Figure 96: OpenStack Create An Image Window

Create An Image

Name *

Description

Image Source

Image Location ▼

Image Location ?

http://example.com/image.iso

Format *

Select Format ▼

Architecture

Minimum Disk (GB) ?

Minimum RAM (MB) ?

☐ Public

☐ Protected

Description:

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

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

s018515

Cancel

Create Image

4. Complete the fields to specify your image. [Table 32 on page 302](#) describes each of the fields on the window.



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

Table 32: Create an Image Fields

Field	Description
Name	Enter a name for this image.

Table 32: Create an Image Fields (*continued*)

Field	Description
Description	Enter a description for the image.
Image Source	<p>Select Image File or Image Location.</p> <p>If you select Image File, you are prompted to browse to the local location of the file.</p>
Image Location	Enter an external HTTP URL from which to load the image. The URL must be a valid and direct URL to the image binary. URLs that redirect or serve error pages result in unusable images.
Format	<p>Required field. Select the format of the image from a list:</p> <p>AKI– Amazon Kernel Image AMI– Amazon Machine Image ARI– Amazon Ramdisk Image ISO– Optical Disk Image QCOW2– QEMU Emulator Raw– An unstructured image format VDI– Virtual Disk Image VHD– Virtual Hard Disk VMDK– Virtual Machine Disk</p>
Architecture	Enter the architecture.
Minimum Disk (GB)	Enter the minimum disk size required to boot the image. If you do not specify a size, the default is 0 (no minimum).
Minimum Ram (MB)	Enter the minimum RAM required to boot the image. If you do not specify a size, the default is 0 (no minimum).
Public	Select this check box if this is a public image. Leave unselected for a private image.
Protected	Select this check box for a protected image.

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

Related Documentation

- *Launching a Virtual Machine (Instance)*

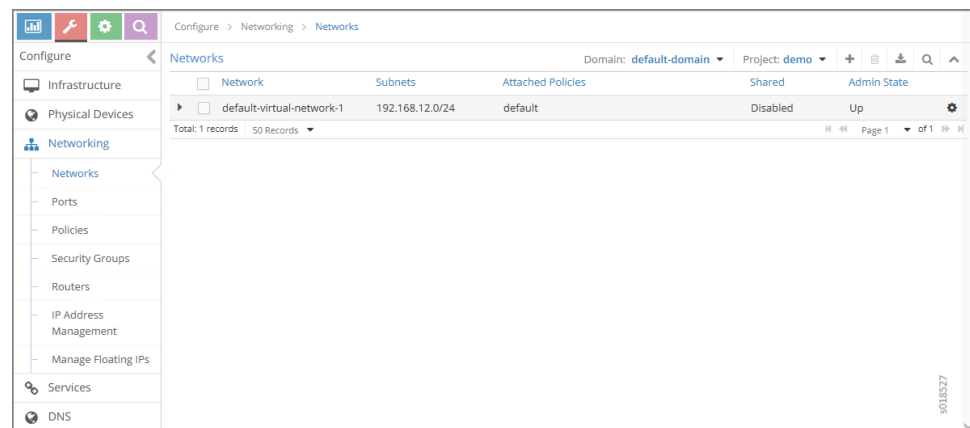
Creating a Floating IP Address Pool

A floating IP address is an IP address (typically public) that can be dynamically assigned to a running virtual instance.

To configure floating IP address pools in project networks in Contrail, then allocate floating IP addresses from the pool to virtual machine instances in other virtual networks:

1. Select **Configure > Networking > Networks**; see [Figure 97 on page 304](#). Make sure your project is the active project in the upper right.

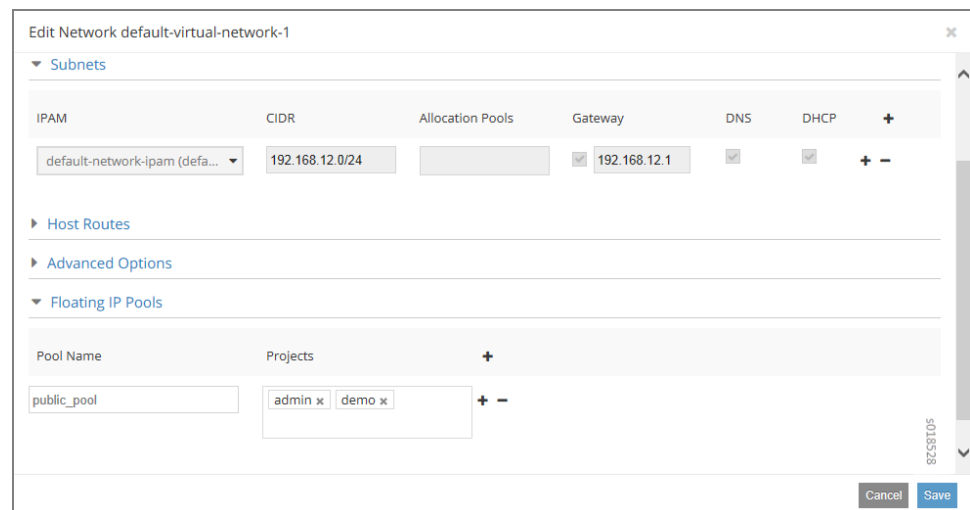
Figure 97: Configure > Networking > Networks



2. Click the network you want to associate with a floating IP pool, then in the **Action** column, click the action icon and select **Edit**.

The **Edit Network** window for the selected network is displayed; see [Figure 98 on page 304](#).

Figure 98: Edit Network



3. In the **Floating IP Pools** section, click the **Pool Name** field, enter a name for your floating IP pool, and click the + (plus sign) to add the IP pool to the table below the field.
 - Multiple floating IP pools can be created at the same time.
 - A floating IP pool can be associated with multiple projects.
4. Click **Save** to create the floating IP address pool, or click **Cancel** to remove your work and start over.

Using Security Groups with Virtual Machines (Instances)

- [Security Groups Overview on page 305](#)
- [Creating Security Groups and Adding Rules on page 305](#)

Security Groups Overview

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

Creating Security Groups and Adding Rules

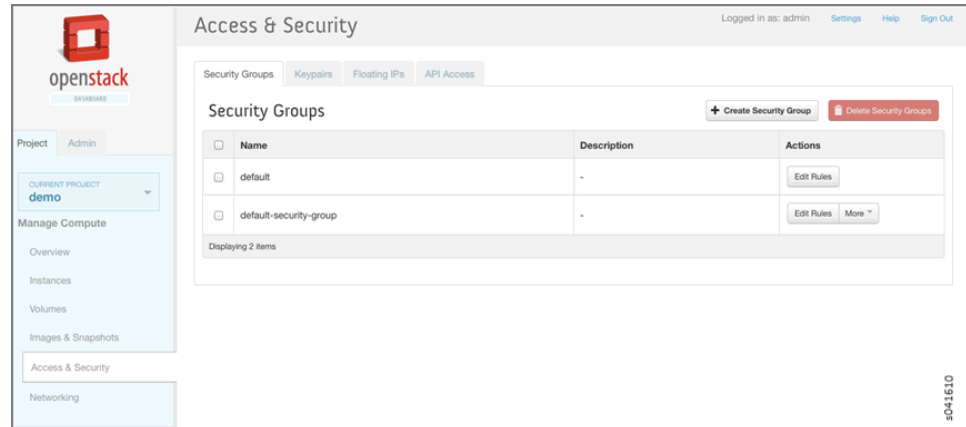
A default security group is created for each project. You can add security rules to the default security group and you can create additional security groups and add rules to them. The security groups are then associated with a VM, when the VM is launched or at a later date.

To add rules to a security group:

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

Any existing security groups are listed under the **Security Groups** tab, including the default security group; see [Figure 99 on page 306](#).

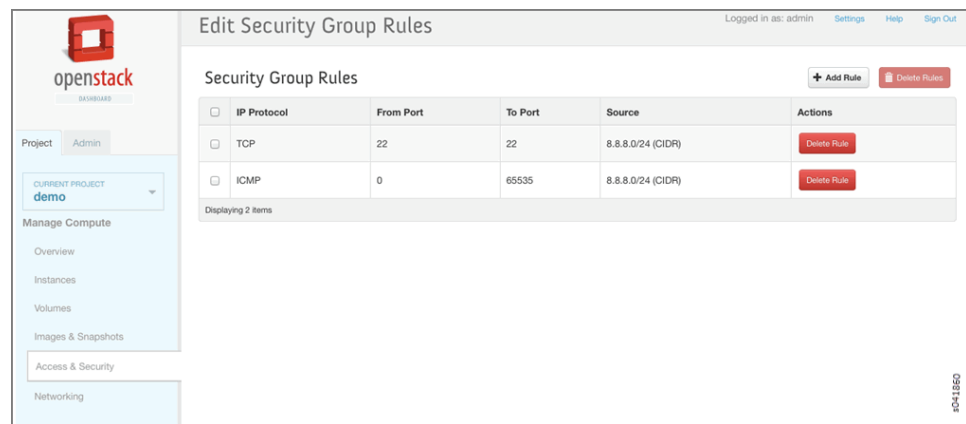
Figure 99: Security Groups



2. Select the **default-security-group** and click **Edit Rules** in the **Actions** column.

The **Edit Security Group Rules** window is displayed; see [Figure 100 on page 306](#). Any rules already associated with the security group are listed.

Figure 100: Edit Security Group Rules



3. Click **Add Rule** to add a new rule; see [Figure 101 on page 307](#).

Figure 101: Add Rule

Add Rule

IP Protocol

Type

Code

Source

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

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

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

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

Table 33: Add Rule Fields

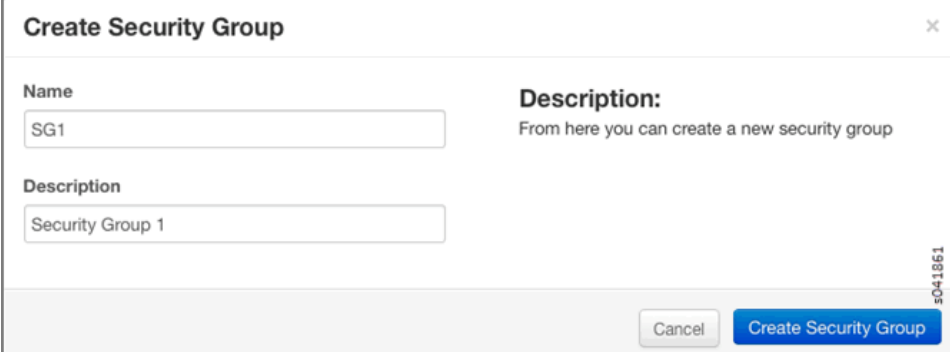
Column	Description
IP Protocol	Select the IP protocol to apply for this rule: TCP, UDP, ICMP.
From Port	Select the port from which traffic originates to apply this rule. For TCP and UDP, enter a single port or a range of ports. For ICMP rules, enter an ICMP type code.
To Port	The port to which traffic is destined that applies to this rule, using the same options as in the From Port field.
Source	Select the source of traffic to be allowed by this rule. Specify subnet—the CIDR IP address or address block of the inter-domain source of the traffic that applies to this rule, or you can choose security group as source. Selecting security group as source allows any other instance in that security group access to any other instance via this rule.

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

The **Create Security Group** window is displayed; see [Figure 102 on page 308](#).

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

Figure 102: Create Security Group



Create Security Group [X]

Name

Description

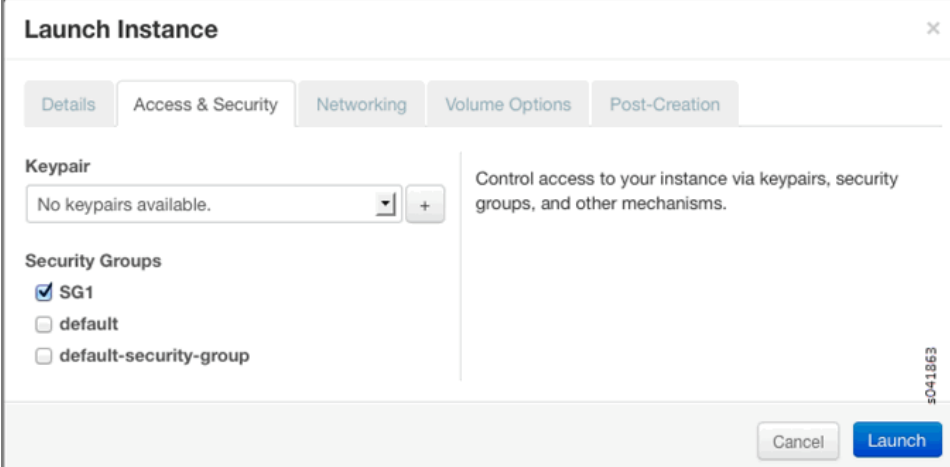
Description:
 From here you can create a new security group

Cancel Create Security Group

- When an instance is launched, there is an opportunity to associate a security group; see [Figure 103 on page 308](#).

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

Figure 103: Associate Security Group at Launch Instance



Launch Instance [X]

Details Access & Security Networking Volume Options Post-Creation

Keypair
 +

Security Groups
☒ SG1
☐ default
☐ default-security-group

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

Cancel Launch

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

Support for IPv6 Networks in Contrail

Starting with Contrail Release 2.0, support for IPv6 overlay networks is provided.

- [Overview: IPv6 Networks in Contrail on page 309](#)
- [Creating IPv6 Virtual Networks in Contrail on page 309](#)
- [Adding IPv6 Peers on page 311](#)

Overview: IPv6 Networks in Contrail

The following features are supported for IPv6 networks and overlay. The underlay network must be IPv4.

- Virtual machines with IPv6 and IPv4 interfaces
- Virtual machines with IPv6-only interfaces
- DHCPv6 and neighbor discovery
- Policy and Security groups
- IPv6 flow set up, tear down, and aging
- Flow set up and tear down based on TCP state machine
- Protocol-based flow aging
- Fat flow
- Allowed address pair configuration with IPv6 addresses
- IPv6 service chaining
- Equal Cost Multi-Path (ECMP)
- Connectivity with gateway (MX Series device)
- Virtual Domain Name Services (vDNS), name-to-IPv6 address resolution
- User-Visible Entities (UVEs)

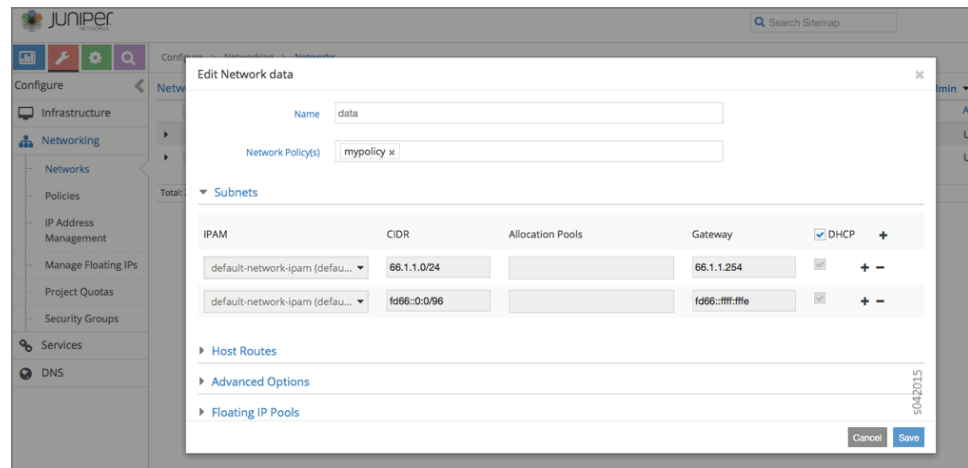
NOT present is support for the following:

- Source Network Address Translation (SNAT)
- Load Balancing as a Service (LBaaS)
- IPv6 fragmentation
- Floating IP
- Link-local and metadata services
- Diagnostics for IPv6
- Contrail Device Manager
- Virtual customer premises equipment (vCPE)

Creating IPv6 Virtual Networks in Contrail

You can create an IPv6 virtual network from the Contrail user interface in the same way you create an IPv4 virtual network. When you create a new virtual network by selecting

Configure > Networking > Networks, the Edit fields accept IPv6 addresses, as shown in the following image.



Address Assignments

When virtual machines are launched with an IPv6 virtual network created in the Contrail user interface, the virtual machine interfaces get assigned addresses from all the families configured in the virtual network.

The following is a sample of IPv6 instances with address assignments, as listed in the OpenStack Horizon user interface.

Instances										
Instances										
Instance Name	Image Name	IP Address	Size	Keypair	Status	Task	Power State	Uptime	Actions	
Test-6b41281-ada9-415c-8609-bcd89578e03	ubuntu-jd4f	data 66.1.1.251 fd66::ffff:fe00 vn-jd4f 76.1.1.252	m1.medium 4GB RAM 2 VCPU 40.0GB Disk	-	Active	None	Running	4 days, 9 hours	Create Snapshot	More
Test-7a3b7f0b-e5a8-48a3-9346-29079e1ab0ba	ubuntu-jd4f	data 66.1.1.250 fd66::ffff:fe00 vn-jd4f 76.1.1.250	m1.medium 4GB RAM 2 VCPU 40.0GB Disk	-	Active	None	Running	4 days, 9 hours	Create Snapshot	More
Test-663309b7-1785-4cc4-9ecd-19025ec04ee5	ubuntu-jd4f	data 66.1.1.245 fd66::ffff:fe00 vn-jd4f 76.1.1.244	m1.medium 4GB RAM 2 VCPU 40.0GB Disk	-	Active	None	Running	4 days, 9 hours	Create Snapshot	More
Test-a20a46c7-5cd5-447e-8894-c794aa820ab	ubuntu-jd4f	data 66.1.1.252 fd66::ffff:fe00 vn-jd4f 76.1.1.251	m1.medium 4GB RAM 2 VCPU 40.0GB Disk	-	Active	None	Running	4 days, 9 hours	Create Snapshot	More
Test-43343608-4055-47a5-9346-5a81f5b2197	ubuntu-jd4f	data 66.1.1.247 fd66::ffff:fe00 vn-jd4f 76.1.1.247	m1.medium 4GB RAM 2 VCPU 40.0GB Disk	-	Active	None	Running	4 days, 9 hours	Create Snapshot	More

Enabling DHCPv6 In Virtual Machines

To allow IPv6 address assignment using DHCPv6, the virtual machine network interface configuration must be updated appropriately.

For example, to enable DHCPv6 for Ubuntu-based virtual machines, add the following line in the `/etc/network/interfaces` file:

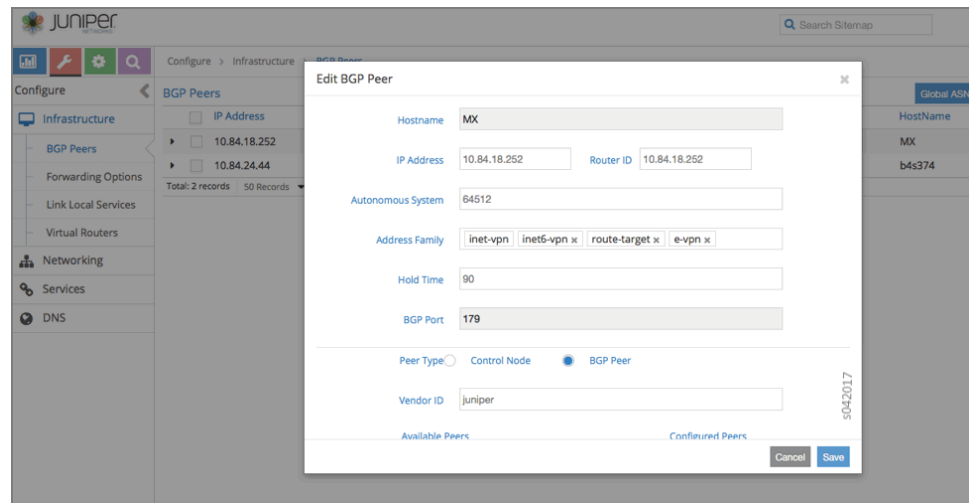
```
iface eht0 inet6 dhcp
```

Also, **dhclient -6** can be run from within the virtual machine to get IPv6 addresses using DHCPv6.

Adding IPv6 Peers

The procedure to add an IPv6 BGP peer in Contrail is similar to adding an IPv4 peer. Select **Configure > Infrastructure > BGP Peers**, include **inet6-vpn** in the Address Family list to allow advertisement of IPv6 addresses.

A sample is shown in the following.



NOTE: Additional configuration is required on the peer router to allow inet6-vpn peering.

Configuring EVPN and VXLAN

Contrail supports Ethernet VPNs (EVPN) and Virtual Extensible Local Area Networks (VXLAN).

EVPN is a flexible solution that uses Layer 2 overlays to interconnect multiple edges (virtual machines) within a data center. Traditionally, the data center is built as a flat Layer 2 network with issues such as flooding, limitations in redundancy and provisioning, and high volumes of MAC address learning, which cause churn during node failures. EVPNs are designed to address these issues without disturbing flat MAC connectivity.

In EVPNs, MAC address learning is driven by the control plane, rather than by the data plane, which helps control learned MAC addresses across virtual forwarders, thus avoiding flooding. The forwarders advertise locally learned MAC addresses to the controllers. The controllers use MP-BGP to communicate with peers. The peering of controllers using BGP for EVPN results in better and faster convergence.

With EVPN, MAC learning is confined to the virtual networks to which the virtual machine belongs, thus isolating traffic between multiple virtual networks. In this manner, virtual networks can share the same MAC addresses without any traffic crossover.

Unicast in EVPNs

Unicast forwarding is based on MAC addresses where traffic can terminate on a local endpoint or is encapsulated to reach the remote endpoint. Encapsulation can be MPLS/UDP, MPLS/GRE, or VXLAN.

BUM Traffic in EVPN

Multicast and broadcast traffic is flooded in a virtual network. The replication tree is built by the control plane, based on the advertisements of end nodes (virtual machines) sent by forwarders. Each virtual network has one distribution tree, a method that avoids maintaining multicast states at fabric nodes, so the nodes are unaffected by multicast. The replication happens at the edge forwarders. Per-group subscription is not provided. Broadcast, unknown unicast, and multicast (BUM) traffic is handled the same way, and gets flooded in the virtual network to which the virtual machine belongs.

VXLAN

VXLAN is an overlay technology that encapsulates MAC frames into a UDP header at Layer 2. Communication is established between two virtual tunnel endpoints (VTEPs). VTEPs encapsulate the virtual machine traffic into a VXLAN header, as well as strip off the encapsulation. Virtual machines can only communicate with each other when they belong to the same VXLAN segment. A 24-bit virtual network identifier (VNID) uniquely identifies the VXLAN segment. This enables having the same MAC frames across multiple VXLAN segments without traffic crossover. Multicast in VXLAN is implemented as Layer 3 multicast, in which endpoints subscribe to groups.

Design Details of EVPN and VXLAN

In Contrail Release 1.03 and later, EVPN is enabled by default. The supported forwarding modes include:

- Fallback bridging—IPv4 traffic lookup is performed using the IP FIB. All non-IPv4 traffic is directed to a MAC FIB.
- Layer 2-only— All traffic is forwarded using a MAC FIB lookup.

You can configure the forwarding mode individually on each virtual network.

EVPN is used to share MAC addresses across different control planes in both forwarding models. The result of a MAC address lookup is a next hop, which, similar to IP forwarding, points to a local virtual machine or a tunnel to reach the virtual machine on a remote server. The tunnel encapsulation methods supported for EVPN are MPLSoGRE, MPLSoUDP, and VXLAN. The encapsulation method selected is based on a user-configured priority.

In VXLAN, the VNID is assigned uniquely for every virtual network carried in the VXLAN header. The VNID uniquely identifies a virtual network. When the VXLAN header is received

from the fabric at a remote server, the VNID lookup provides the VRF of the virtual machine. This VRF is used for the MAC lookup from the inner header, which then provides the destination virtual machine.

Non-IP multicast traffic uses the same multicast tree as for IP multicast (255.255.255.255). The multicast is matched against the all-broadcast prefix in the bridging table (FF:FF:FF:FF:FF:FF). VXLAN is not supported for IP/non-IP multicast traffic.

The following table summarizes the traffic and encapsulation types supported for EVPN.

		Encapsulation		
		MPLS-GRE	MPLS-UDP	VXLAN
Traffic Type	IP unicast	Yes	Yes	No
	IP-BUM	Yes	Yes	No
	non IP unicast	Yes	Yes	Yes
	non IP-BUM	Yes	Yes	No

- [Configuring the VXLAN Identifier Mode on page 313](#)
- [Configuring Forwarding on page 315](#)
- [Configuring the VXLAN Identifier on page 316](#)
- [Configuring Encapsulation Methods on page 317](#)

Configuring the VXLAN Identifier Mode

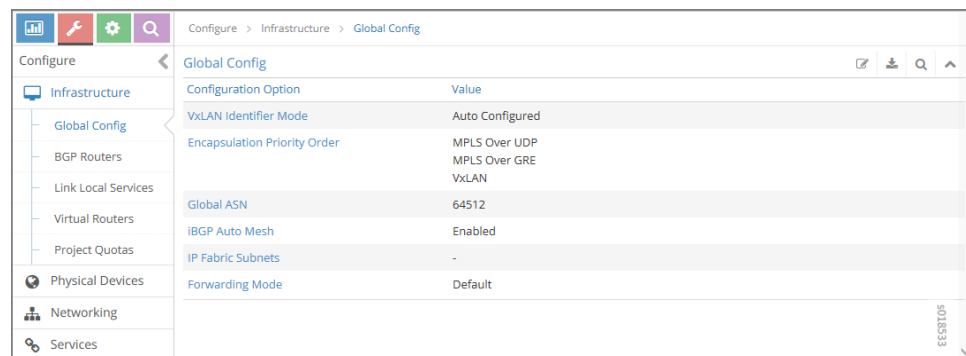
You can configure the global VXLAN identifier mode to select an auto-generated VNID or a user-generated VXLAN ID, either through the Contrail Web UI or by modifying a python file.

To configure the global VXLAN identifier mode:

1. From the Contrail Web UI, select **Configure > Infrastructure > Global Config**.

The Global Config options and values are displayed in the Global Config window.

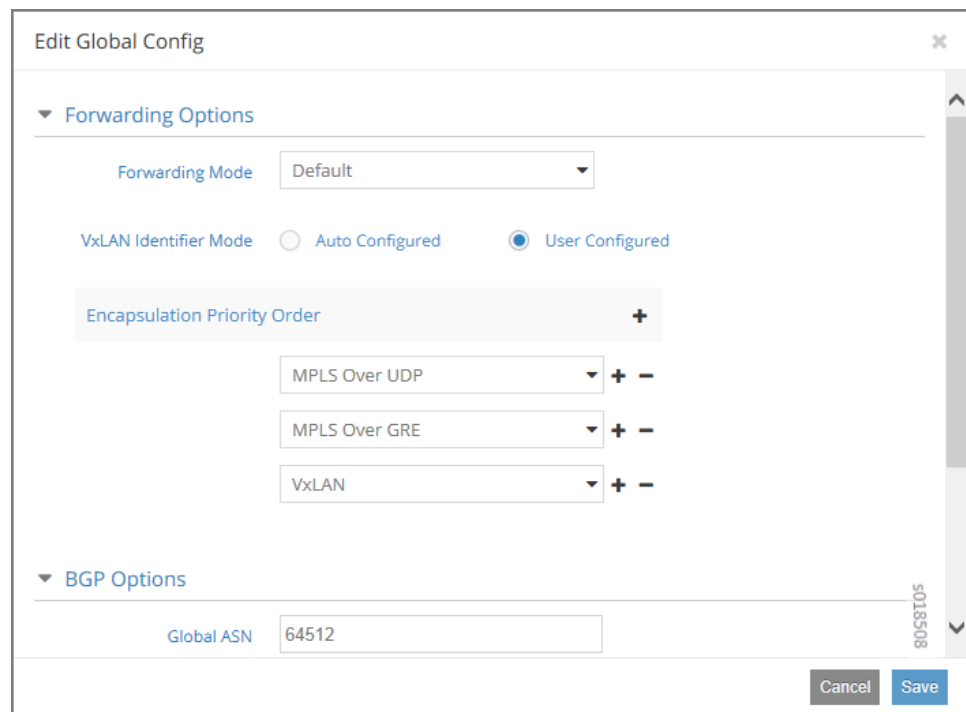
Figure 104: Global Config Window for VXLAN ID



2. Click the edit icon .

The Edit Global Config window is displayed as shown in [Figure 105 on page 314](#).

Figure 105: Edit Global Config Window for VXLAN Identifier Mode



3. Select one of the following:
 - **Auto Configured**— The VXLAN identifier is automatically assigned for the virtual network.
 - **User Configured**— You must provide the VXLAN identifier for the virtual network.



NOTE: When **User Configured** is selected, if you do not provide an identifier, then VXLAN encapsulation *is not used* and the mode falls back to MPLS.


Alternatively, you can set the VXLAN identifier mode by using Python to modify the `/opt/contrail/utls/encap.py` file as follows:

```
python encap.py <add | update | delete> <username> <password> <tenant_name> <config_node_ip>
```

Configuring Forwarding

In Contrail, the default forwarding mode is enabled for fallback bridging (IP FIB and MAC FIB). The mode can be changed, either through the Contrail Web UI or by using python provisioning commands.

To change the forwarding mode:

1. From the Contrail Web UI, select **Configure > Networking > Networks**.
2. Select the virtual network that you want to change the forwarding mode for.
3. Click the gear icon  and select **Edit**.

The Edit Network window is displayed as shown in [Figure 106 on page 315](#).

Figure 106: Edit Network Window

IPAM	CIDR	Allocation Pools	Gateway	DNS	DHCP
TestProjectC5Ca5C-ipam655...	31.222.172.0/24		<input checked="" type="checkbox"/> 31.222.172.1	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

Under the Advanced Options select the forwarding mode from the following choices:

- Select **Default** to enable the default forwarding mode.
- Select **L2 and L3** to enable IP and MAC FIB (fallback bridging).

- Select **L2 Only** to enable only MAC FIB.
- Select **L3 Only** to enable only IP.



NOTE: The full list of forwarding modes are only displayed if you change entries in the `/usr/src/contrail/contrail-web-core/config/config.global.js` file. For example:

1. To make the **L2** selection available locate the following:

```
config.network = {};  
config.network.L2_enable = false;
```

2. Change the entry to the following:

```
config.network = {};  
config.network.L2_enable = true;
```

3. To make the other selections available, modify the corresponding entries.

4. Save the file and quit the editor.

5. Restart the Contrail Web user interface process (webui).

Alternatively, you can use the following python provisioning command to change the forwarding mode:

```
python provisioning_forwarding_mode --project_fq_name 'defaultdomain:admin' --vn_name  
vn1 --forwarding_mode < l2_l3| l2 >
```

Options:

l2_l3 = Enable IP FIB and MAC FIB (fallback bridging)

l2 = Enable MAC FIB only (Layer 2 only)

Configuring the VXLAN Identifier

The VXLAN identifier can be set only if the VXLAN network identifier mode has been set to User Configured. You can then set the VXLAN ID by either using the Contrail Web UI or by using Python commands.

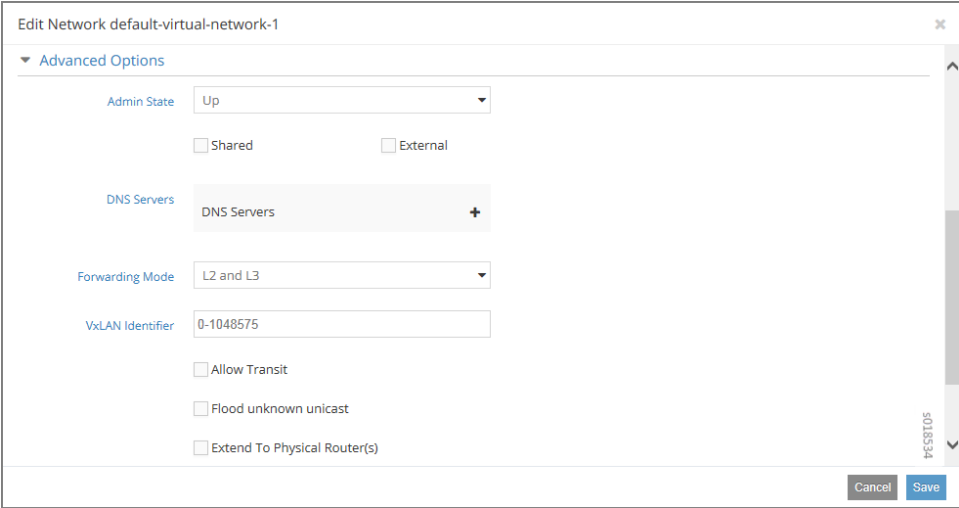
To configure the global VXLAN identifier:

1. From the Contrail Web UI, select **Configure > Networking > Networks**.
2. Select the virtual network that you want to change the forwarding mode for.

3. Click the gear icon  and select **Edit**.

The Edit Network window is displayed. Select the **Advanced Options** as shown in [Figure 107 on page 317](#).

Figure 107: Edit Network Window for VXLAN Identifier



Advanced Options

Admin State: Up

☐ Shared ☐ External

DNS Servers: DNS Servers +

Forwarding Mode: L2 and L3

VxLAN Identifier: 0-1048575

☐ Allow Transit

☐ Flood unknown unicast

☐ Extend To Physical Router(s)

Cancel Save

4. Type the VXLAN identifier.

5. Click **Save**.

Alternatively, you can use the following Python provisioning command to configure the VXLAN identifier:

```
python provisioning_forwarding_mode --project_fq_name 'defaultdomain: admin' --vn_name vn1 --forwarding_mode < vxlan_id >
```


Configuring Encapsulation Methods

The default encapsulation mode for EVPN is MPLS over UDP. All packets on the fabric are encapsulated with the label allocated for the virtual machine interface. The label encoding and decoding is the same as for IP forwarding. Additional encapsulation methods supported for EVPN include MPLS over GRE and VXLAN. MPLS over UDP is different from MPLS over GRE only in the method of tunnel header encapsulation.

VXLAN has its own header and uses a VNID label to carry the traffic over the fabric. A VNID is assigned with every virtual network and is shared by all virtual machines in the virtual network. The VNID is mapped to the VRF of the virtual network to which it belongs.

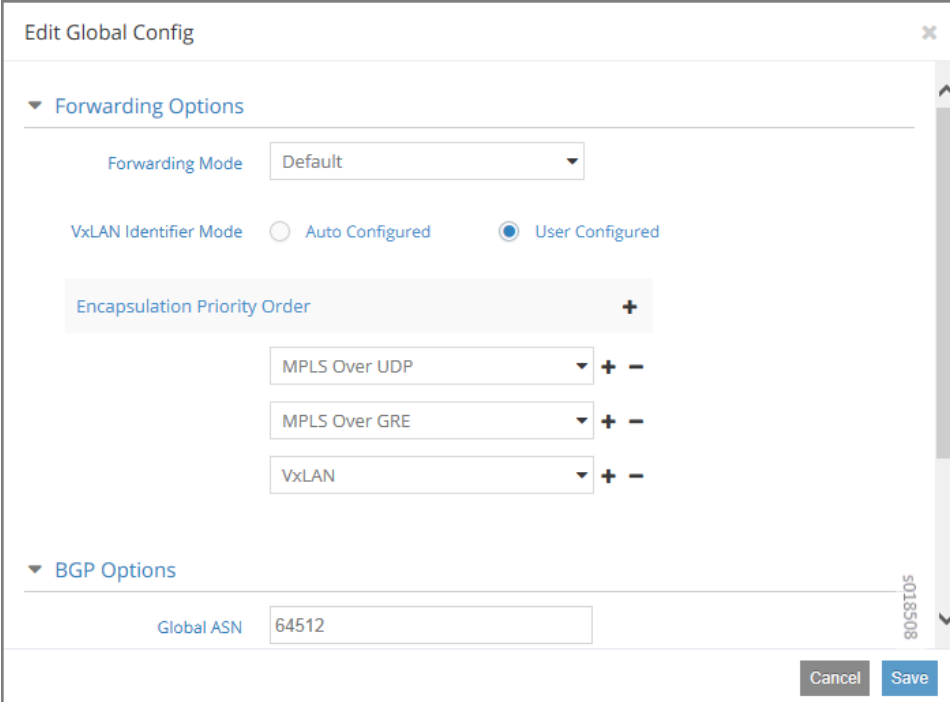
The priority order in which to apply encapsulation methods is determined by the sequence of methods set either from the Contrail Web UI or in the **encap.py** file.

To configure the global VXLAN identifier mode:

- From the Contrail Web UI, select **Configure > Infrastructure > Global Config**.
- The Global Config options are displayed.
- Click the edit icon  .

The Edit Global Config window is displayed as shown in [Figure 108 on page 318](#).

Figure 108: Edit Global Config Window for Encapsulation Priority Order



The screenshot shows the 'Edit Global Config' window. It has a title bar with a close button. The main content is divided into sections. The 'Forwarding Options' section is expanded, showing 'Forwarding Mode' set to 'Default'. Below it, 'VxLAN Identifier Mode' has two radio buttons: 'Auto Configured' (unselected) and 'User Configured' (selected). Underneath is the 'Encapsulation Priority Order' section, which has a '+' button to add more priorities. It currently lists three priorities: 'MPLS Over UDP', 'MPLS Over GRE', and 'VxLAN', each with '+' and '-' buttons to adjust its position. The 'BGP Options' section is also expanded, showing 'Global ASN' set to '64512'. At the bottom right, there are 'Cancel' and 'Save' buttons. A vertical scrollbar is visible on the right side of the window.

Under Encapsulation Priority Order select one of the following:

- **MPLS over UDP**
- **MPLS over GRE**
- **VxLAN**

Click the + plus symbol to the right of the first priority to add a second priority or third priority.

Use the following procedure to change the default encapsulation method to VXLAN by editing the `encap.py` file.



NOTE: VXLAN is *only* supported for EVPN unicast. It is not supported for IP traffic or multicast traffic. VXLAN priority and presence in the `encap.py` file or configured in the Web UI is ignored for traffic not supported by VXLAN.

To set the priority of encapsulation methods to VXLAN:

1. Modify the `encap.py` file found in the `/opt/contrail/utils/` directory.

The default encapsulation line is:

```
encap_obj=EncapsulationPrioritiesType(encapsulation=['MPLSoUDP','MPLSoGRE'])
```

Modify the line to:

```
encap_obj=EncapsulationPrioritiesType(encapsulation=['VXLAN',
'MPLSoUDP','MPLSoGRE'])
```

2. After the status is modified, execute the following script:

```
python encap_set.py <add|update|delete> <username> <password> <tenant_name>
<config_node_ip>
```

The configuration is applied globally for all virtual networks.

CHAPTER 13

Example of Deploying a Multi-Tier Web Application Using Contrail

- [Example: Deploying a Multi-Tier Web Application on page 321](#)
- [Sample Network Configuration for Devices for Simple Tiered Web Application on page 327](#)

Example: Deploying a Multi-Tier Web Application

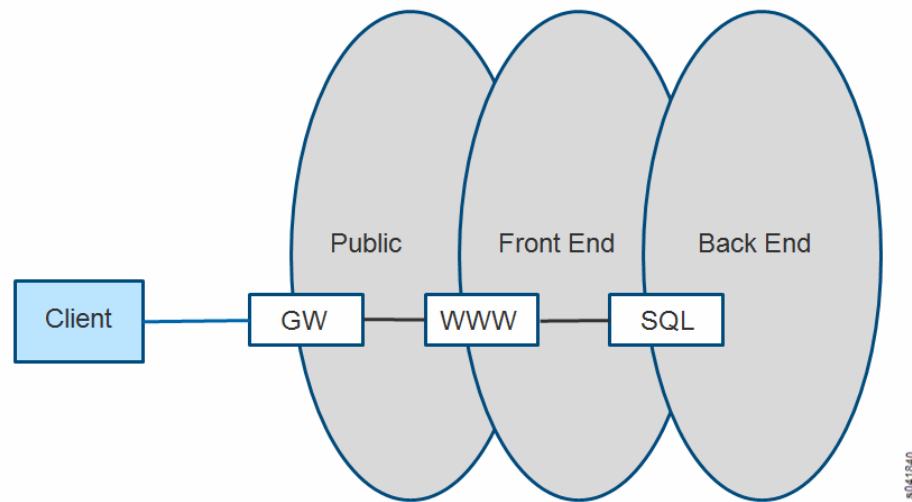
- [Multi-Tier Web Application Overview on page 321](#)
- [Example: Setting Up Virtual Networks for a Simple Tiered Web Application on page 322](#)
- [Verifying the Multi-Tier Web Application on page 324](#)
- [Sample Addressing Scheme for Simple Tiered Web Application on page 325](#)
- [Sample Physical Topology for Simple Tiered Web Application on page 326](#)
- [Sample Physical Topology Addressing on page 326](#)

Multi-Tier Web Application Overview

A common requirement for a cloud tenant is to create a tiered web application in leased cloud space. The tenant enjoys the favorable economics of a private IT infrastructure within a shared services environment. The tenant seeks speedy setup and simplified operations.

The following example shows how to set up a simple tiered web application using Contrail. The example has a web server that a user accesses by means of a public floating IP address. The front-end web server gets the content it serves to customers from information stored in a SQL database server that resides on a back-end network. The web server can communicate directly with the database server without going through any gateways. The public (or client) can only communicate to the web server on the front-end network. The client is not allowed to communicate directly with any other parts of the infrastructure. See [Figure 109 on page 322](#).

Figure 109: Simple Tiered Web Use Case



Example: Setting Up Virtual Networks for a Simple Tiered Web Application

This example provides basic steps for setting up a simple multi-tier network application. Basic creation steps are provided, along with links to the full explanation for each of the creation steps. Refer to the links any time you need more information about completing a step.

1. Working with a system that has the Contrail software installed and provisioned, create a project named **demo**.

For more information; see [“Creating Projects in OpenStack for Configuring Tenants in Contrail” on page 294](#).

2. In the **demo** project, create three virtual networks:

- a. A network named **public** with IP address **10.84.41.0/24**

This is a special use virtual network for floating IP addresses— it is assigned an address block from the public floating address pool that is assigned to each web server. The assigned block is the only address block advertised outside of the data center to clients that want to reach the web services provided.

- b. A network named **frontend** with IP address **192.168.1.0/24**

This network is the location where the web server virtual machine instances are launched and attached. The virtual machines are identified with private addresses that have been assigned to this virtual network.

- c. A network named **backend** with IP address **192.168.2.0/24**

This network is the location where the database server virtual machines instances are launched and attached. The virtual machines are identified with private addresses that have been assigned to this virtual network.

For more information; see “Creating a Virtual Network with OpenStack Contrail” on page 299 or “Creating a Virtual Network with Juniper Networks Contrail” on page 296.

3. Create a floating IP pool named **public_pool** for the **public** network within the **demo** project; see Figure 110 on page 323.

Figure 110: Create Floating IP Pool

Edit Network public

Network Name: public

Network Policy(s): Select Policies...

Address Management: default-network... + -

IPAM	IP Block
default-network-ipam	10.84.41.0/24

Floating IP Pools: public_pool + -

Pool Name: admin

Cancel Save

4. Allocate the floating IP pool **public_pool** to the **demo** project; see Figure 111 on page 323.

Figure 111: Allocate Floating IP

Allocate Floating IP

Floating IP Pool: public:public_pool

Cancel Save

5. Verify that the floating IP pool has been allocated; see **Configure > Networking > Allocate Floating IPs**.

For more information; see [“Creating a Floating IP Address Pool”](#) on page 304 and [Allocating a Floating IP Address to a Virtual Machine](#).

6. Create a policy that allows any host to talk to any host using any IP address, protocol, and port, and apply this policy between the **frontend** network and the **backend** network.

This now allows communication between the web servers in the front-end network and the database servers in the back-end network.

For more information; see *Creating a Network Policy—Juniper Networks Contrail*, *Associating a Network to a Policy—Juniper Networks Contrail*, or *Creating a Network Policy—OpenStack Contrail*, and *Associating a Network to a Policy—OpenStack Contrail*.

7. Launch the virtual machine instances that represent the web server and the database server.



NOTE: Your installation might not include the virtual machines needed for the web server and the database server. Contact your account team if you need to download the VMs for this setup.

On the **Instances** tab for this project, select **Launch Instance** and for each instance that you launch, complete the fields to make the following associations:

- Web server VM: select **frontend** network and the policy created to allow communication between **frontend** and **backend** networks. Apply the floating IP address pool to the web server.
- Database server VM: select **backend** network and the policy created to allow communication between **frontend** and **backend** networks.

For more information; see *Launching a Virtual Machine (Instance)*.

Verifying the Multi-Tier Web Application

Verify your web setup.

- To demonstrate this web application setup, go to the client machine, open a browser, and navigate to the address in the **public** network that is assigned to the web server in the **frontend** network.

The result will display the Contrail interface with various data populated, verifying that the web server is communicating with the database server in the **backend** network and retrieving data.

The client machine only has access to the public IP address. Attempts to browse to any of the addresses assigned to the **frontend** network or to the **backend** network should fail.

Sample Addressing Scheme for Simple Tiered Web Application

Use the information in [Table 34 on page 325](#) as a guide for addressing devices in the simple tiered web example.

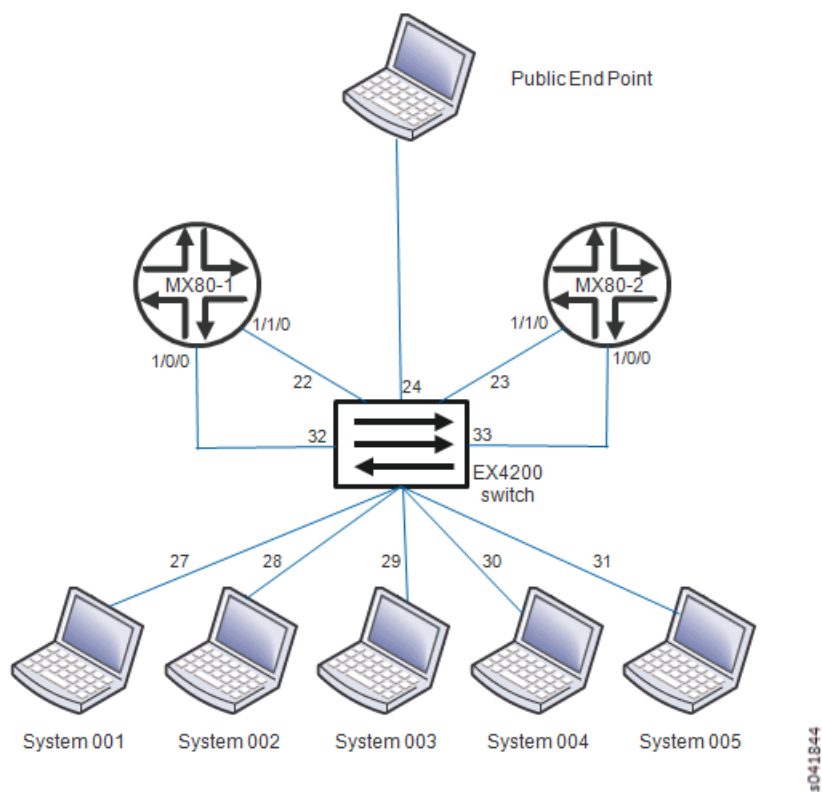
Table 34: Sample Addressing Scheme for Example

System Name	Address Allocation
System001	10.84.11.100
System002	10.84.11.101
System003	10.84.11.102
System004	10.84.11.103
System005	10.84.11.104
MX80-1	10.84.11.253 10.84.45.1 (public connection)
MX80-2	10.84.11.252 10.84.45.2 (public connection)
EX4200	10.84.11.254 10.84.45.254 (public connection) 10.84.63.259 (public connection)
frontend network	192.168.1.0/24
backend network	192.168.2.0/24
public network (floating address)	10.84.41.0/24

Sample Physical Topology for Simple Tiered Web Application

Figure 112 on page 326 provides a guideline diagram for the physical topology for the simple tiered web application example.

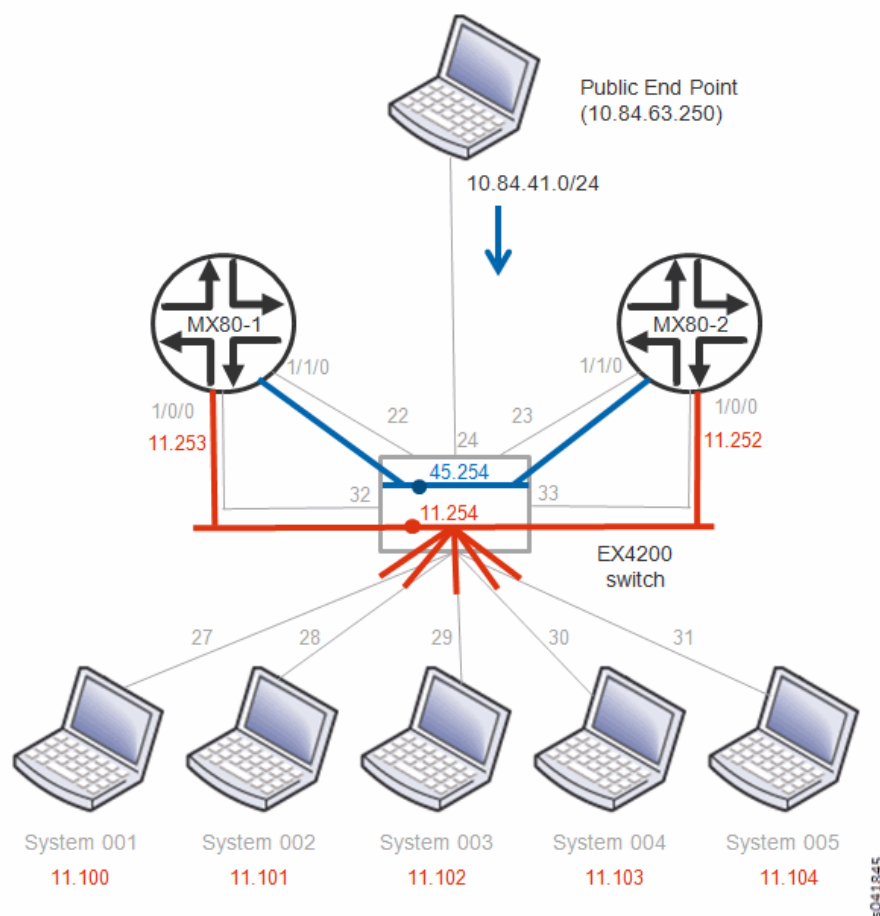
Figure 112: Sample Physical Topology for Simple Tiered Web Application



Sample Physical Topology Addressing

Figure 113 on page 327 provides a guideline diagram for addressing the physical topology for the simple tiered web application example.

Figure 113: Sample Physical Topology Addressing



See Also • [Sample Network Configuration for Devices for Simple Tiered Web Application on page 327](#)

Sample Network Configuration for Devices for Simple Tiered Web Application

This section shows sample device configurations that can be used to create the “[Example: Deploying a Multi-Tier Web Application](#)” on page 321. Configurations are shown for Juniper Networks devices: two MX80s and one EX4200.

MX80-1 Configuration

```
version 12.2R1.3;
system {
  root-authentication {
    encrypted-password "xxxxxxxxx"; ## SECRET-DATA
  }
  services {
    ssh {
```

```
        root-login allow;
    }
}
syslog {
    user * {
        any emergency;
    }
    file messages {
        any notice;
        authorization info;
    }
}
chassis {
    fpc 1 {
        pic 0 {
            tunnel-services;
        }
    }
}
interfaces {
    ge-1/0/0 {
        unit 0 {
            family inet {
                address 10.84.11.253/24;
            }
        }
    }
    ge-1/1/0 {
        description "IP Fabric interface";
        unit 0 {
            family inet {
                address 10.84.45.1/24;
            }
        }
    }
    lo0 {
        unit 0 {
            family inet {
                address 127.0.0.1/32;
            }
        }
    }
}
routing-options {
    static {
        route 0.0.0.0/0 next-hop 10.84.45.254;
    }
    route-distinguisher-id 10.84.11.253;
    autonomous-system 64512;
    dynamic-tunnels {
        setup1 {
            source-address 10.84.11.253;
            gre;
            destination-networks {
                10.84.11.0/24;
            }
        }
    }
}
```



```

    }
  }
}
protocols {
  bgp {
    group mx {
      type internal;
      local-address 10.84.11.253;
      family inet-vpn {
        unicast;
      }
      neighbor 10.84.11.252;
    }
    group contrail-controller {
      type internal;
      local-address 10.84.11.253;
      family inet-vpn {
        unicast;
      }
      neighbor 10.84.11.101;
      neighbor 10.84.11.102;
    }
  }
}
routing-instances {
  customer-public {
    instance-type vrf;
    interface ge-1/1/0.0;
    vrf-target target:64512:10000;
    routing-options {
      static {
        route 0.0.0.0/0 next-hop 10.84.45.254;
      }
    }
  }
}
}

```

MX80-2 Configuration

```

version 12.2R1.3;
system {
  root-authentication {
    encrypted-password "xxxxxxxxx"; ## SECRET-DATA
  }
  services {
    ssh {
      root-login allow;
    }
  }
  syslog {
    user * {
      any emergency;
    }
    file messages {
      any notice;
    }
  }
}

```

```
        authorization info;
    }
}
chassis {
  fpc 1 {
    pic 0 {
      tunnel-services;
    }
  }
}
interfaces {
  ge-1/0/0 {
    unit 0 {
      family inet {
        address 10.84.11.252/24;
      }
    }
  }
  ge-1/1/0 {
    description "IP Fabric interface";
    unit 0 {
      family inet {
        address 10.84.45.2/24;
      }
    }
  }
  lo0 {
    unit 0 {
      family inet {
        address 127.0.0.1/32;
      }
    }
  }
}
routing-options {
  static {
    route 0.0.0.0/0 next-hop 10.84.45.254;
  }
  route-distinguisher-id 10.84.11.252;
  autonomous-system 64512;
  dynamic-tunnels {
    setup1 {
      source-address 10.84.11.252;
      gre;
      destination-networks {
        10.84.11.0/24;
      }
    }
  }
}
protocols {
  bgp {
    group mx {
      type internal;
      local-address 10.84.11.252;
```

```

        family inet-vpn {
            unicast;
        }
        neighbor 10.84.11.253;
    }
    group contrail-controller {
        type internal;
        local-address 10.84.11.252;
        family inet-vpn {
            unicast;
        }
        neighbor 10.84.11.101;
        neighbor 10.84.11.102;
    }
}
}
routing-instances {
    customer-public {
        instance-type vrf;
        interface ge-1/1/0.0;
        vrf-target target:64512:10000;
        routing-options {
            static {
                route 0.0.0.0/0 next-hop 10.84.45.254;
            }
        }
    }
}
}
}

```

EX4200 Configuration

```

system {
    host-name EX4200;
    time-zone America/Los_Angeles;
    root-authentication {
        encrypted-password "xxxxxxxxxxxx"; ## SECRET-DATA
    }
    login {
        class read {
            permissions [ clear interface view view-configuration ];
        }
        user admin {
            uid 2000;
            class super-user;
            authentication {
                encrypted-password "xxxxxxxxxxxx"; ## SECRET-DATA
            }
        }
        user user1 {
            uid 2002;
            class read;
            authentication {
                encrypted-password "xxxxxxxxxxxx"; ## SECRET-DATA
            }
        }
    }
}

```

```
services {
  ssh {
    root-login allow;
  }
  telnet;
  netconf {
    ssh;
  }
  web-management {
    http;
  }
}
syslog {
  user * {
    any emergency;
  }
  file messages {
    any notice;
    authorization info;
  }
  file interactive-commands {
    interactive-commands any;
  }
}
chassis {
  aggregated-devices {
    ethernet {
      device-count 64;
    }
  }
}
```

CHAPTER 14

Configuring Services

- [Configuring DNS Servers on page 333](#)
- [Distributed Service Resource Allocation with Containerized Contrail on page 342](#)
- [Support for Multicast on page 349](#)
- [Using Static Routes with Services on page 351](#)
- [Configuring Metadata Service on page 355](#)

Configuring DNS Servers

- [DNS Overview on page 333](#)
- [Defining Multiple Virtual Domain Name Servers on page 334](#)
- [IPAM and Virtual DNS on page 334](#)
- [DNS Record Types on page 335](#)
- [Configuring DNS Using the Interface on page 336](#)
- [Configuring DNS Using Scripts on page 341](#)

DNS Overview

Domain Name System (DNS) is the standard protocol for resolving domain names into IP addresses so that traffic can be routed to its destination. DNS provides the translation between human-readable domain names and their IP addresses. The domain names are defined in a hierarchical tree, with a root followed by top-level and next-level domain labels.

A DNS server stores the records for a domain name and responds to queries from clients based on these records. The server is authoritative for the domains for which it is configured to be the name server. For other domains, the server can act as a caching server, fetching the records by querying other domain name servers.

The following are the key attributes of domain name service in a virtual world:

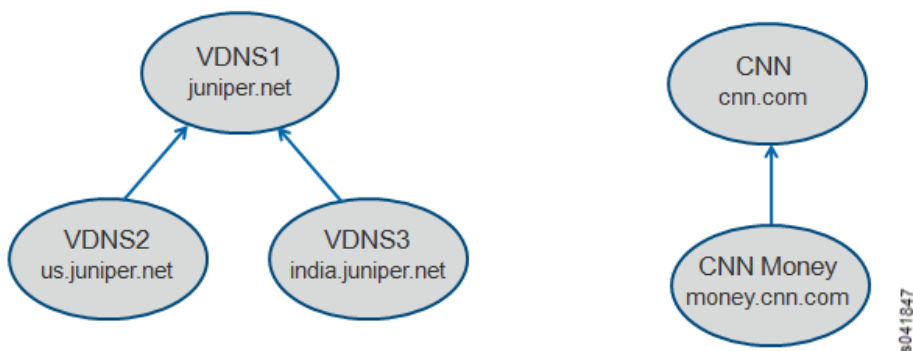
- It should be possible to configure multiple domain name servers to provide name resolution service for the virtual machines spawned in the system.
- It should be possible to configure the domain name servers to form DNS server hierarchies required by each tenant.

- The hierarchies can be independent and completely isolated from other similar hierarchies present in the system, or they can provide naming service to other hierarchies present in the system.
- DNS records for the virtual machines spawned in the system should be updated dynamically when a virtual machine is created or destroyed.
- The service should be scalable to handle an increase in servers and the resulting increased numbers of virtual machines and DNS queries handled in the system.

Defining Multiple Virtual Domain Name Servers

Contrail provides the flexibility to define multiple virtual domain name servers under each domain in the system. Each virtual domain name server is an authoritative server for the DNS domain configured. [Figure 114 on page 334](#) shows examples of virtual DNS servers defined in **default-domain**, providing the name service for the DNS domains indicated.

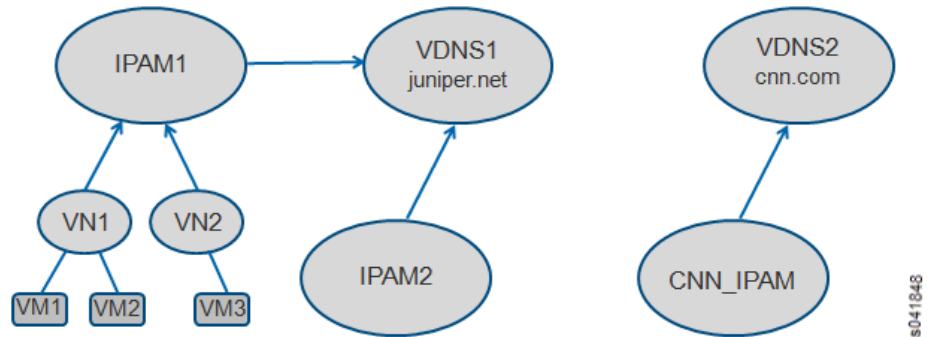
Figure 114: DNS Servers Examples



IPAM and Virtual DNS

Each IP address management (IPAM) service in the system can refer to one of the virtual DNS servers configured. The virtual networks and virtual machines spawned are associated with the DNS domain specified in the corresponding IPAM. When the VMs are configured with DHCP, they receive the domain assignment in the DHCP **domain-name** option. Examples are shown in [Figure 115 on page 335](#)

Figure 115: IPAM and Virtual DNS



DNS Record Types

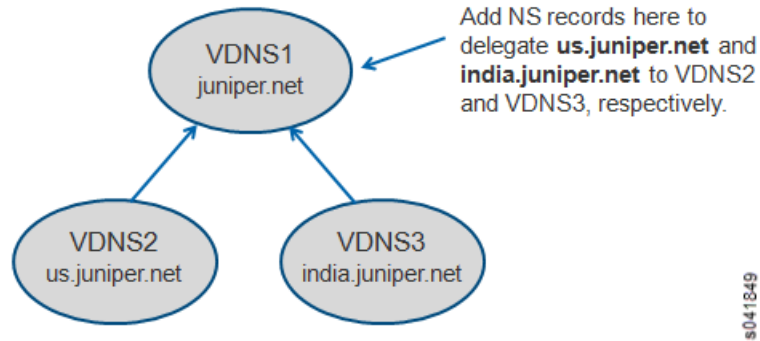
DNS records can be added statically. DNS record types **A**, **CNAME**, **PTR**, and **NS** are currently supported in the system. Each record includes the type, class (IN), name, data, and TTL values. See [Table 35 on page 335](#) for descriptions of the record types.

Table 35: DNS Record Types Supported

DNS Record Type	Description
A	Used for mapping hostnames to IPv4 addresses. Name refers to the name of the virtual machine, and data is the IPv4 address of the virtual machine.
CNAME	Provides an alias to a name. Name refers to the name of the virtual machine, and data is the new name (alias) for the virtual machine.
PTR	A pointer to a record, it provides reverse mapping from an IP address to a name. Name refers to the IP address, and data is the name for the virtual machine. The address in the PTR record should be part of a subnet configured for a VN within one of the IPAMs referring to this virtual DNS server.
NS	Used to delegate a subdomain to another DNS server. The DNS server could be another virtual DNS server defined in the system or the IP address of an external DNS server reachable via the infrastructure. Name refers to the subdomain being delegated, and data is the name of the virtual DNS server or IP address of an external server.

[Figure 116 on page 336](#) shows an example usage for the DNS record type of **NS**.

Figure 116: Example Usage for NS Record Type



Configuring DNS Using the Interface

DNS can be configured by using the user interface or by using scripts. The following procedure shows how to configure DNS through the Juniper Networks Contrail interface.

1. Access **Configure > DNS > Servers** to create or delete virtual DNS servers and records.

The **Configure DNS Records** page appears; see [Figure 117 on page 336](#).

Figure 117: Configure DNS Records

Configure > DNS > Servers Search

Configure DNS Records default-domain admin

Configure Virtual DNS Create Delete

Virtual DNS Name	DNS Domain Name	Next DNS Server
No Data Found		

DNS Records Associated IPAMs

DNS Records of {{dnsname}} Add Record Delete

Name	Type : Data	TTL (secs)	Class

s041850

2. To add a new DNS server, click the **Create** button.

Enter DNS server information in the **Add DNS** window; see [Figure 118 on page 337](#)

Figure 118: Add DNS

Complete the fields for the new server; see [Table 36 on page 337](#).

Table 36: Add DNS Fields

Field	Description
Server Name	Enter a name for this server.
Domain Name	Enter the name of the domain for this server.
Time To Live	Enter the TTL in seconds.
Next DNS Server	Select from a list the name of the next DNS server to process DNS requests if they cannot be processed at this server, or None .
Load Balancing Order	Select the load-balancing order from a list— Random , Fixed , Round Robin . When a name has multiple records matching, the configured record order determines the order in which the records are sent in the response. Select Random to have the records sent in random order. Select Fixed to have records sent in the order of creation. Select Round Robin to have the record order cycled for each request to the record.
OK	Click OK to create the record.
Cancel	Click Cancel to clear the fields and start over.

- To add a new DNS record, from the **Configure DNS Records** page, click the **Add Record** button in the lower right portion of the screen.

The **Add DNS Record** window appears; see [Figure 119 on page 338](#).

Figure 119: Add DNS Record

4. Complete the fields for the new record; see [Table 37 on page 338](#).

Table 37: Add DNS Record Fields

Field	Description
Record Name	Enter a name for this record.
Type	Select the record type from a list— A , CNAME , PTR , NS .
IP Address	Enter the IP address for the location for this record.
Class	Select the record class from a list— IN is the default.
Time To Live	Enter the TTL in seconds.
OK	Click OK to create the record.
Cancel	Click Cancel to clear the fields and start over.

5. To associate an IPAM to a virtual DNS server, from the **Configure DNS Records** page, select the **Associated IPAMs** tab in the lower right portion of the screen and click the **Edit** button.

The **Associate IPAMs to DNS** window appears; see [Figure 120 on page 339](#).

Figure 120: Associate IPAMs to DNS

Complete the IPAM associations, using the field descriptions in [Table 38 on page 339](#).

Table 38: Associate IPAMs to DNS Fields

Field	Description
Associate to All IPAMs	Select this box to associate the selected DNS server to all available IPAMs.
Available IPAMs	This column displays the currently available IPAMs.
Associated IPAMs	This column displays the IPAMs currently associated with the selected DNS server.
>>	Use this button to associate an available IPAM to the selected DNS server, by selecting an available IPAM in the left column and clicking this button to move it to the Associated IPAMs column. The selected IPAM is now associated with the selected DNS server.
<<	Use this button to disassociate an IPAM from the selected DNS server, by selecting an associated IPAM in the right column and clicking this button to move it to the left column (Available IPAMs). The selected IPAM is now disassociated from the selected DNS server.
OK	Click OK to commit the changes indicated in the window.
Cancel	Click Cancel to clear all entries and start over.

- Use the **IP Address Management** page (**Configure > Networking > IP Address Management**); see [Figure 121 on page 340](#)) to configure the DNS mode for any DNS server and to associate an IPAM to DNS servers of any mode or to tenants' IP addresses.

Figure 121: Configure IP Address Management

- To associate an IPAM to a virtual DNS server or to tenant's IP addresses, at the **IP Address Management** page, select the network associated with this IPAM, then click the **Action** button in the last column, and click **Edit**.

The **Edit IP Address Management** window appears; see [Figure 122 on page 340](#).

Figure 122: DNS Server

- In the first field, select the **DNS Method** from a list (**None**, **Default DNS**, **Tenant DNS**, **Virtual DNS**; see [Table 39 on page 340](#)).

Table 39: DNS Modes

DNS Mode	Description
None	Select None when no DNS support is required for the VMs.
Default	In default mode, DNS resolution for VMs is performed based on the name server configuration in the server infrastructure. The subnet default gateway is configured as the DNS server for the VM, and the DHCP response to the VM has this DNS server option. DNS requests sent by a VM to the default gateway are sent to the name servers configured on the respective compute nodes. The responses are sent back to the VM.

Table 39: DNS Modes (*continued*)

DNS Mode	Description
Tenant	Configure this mode when a tenant wants to use its own DNS servers. Configure the list of servers in the IPAM. The server list is sent in the DHCP response to the VM as DNS servers. DNS requests sent by the VMs are routed the same as any other data packet based on the available routing information.
Virtual DNS	Configure this mode to support virtual DNS servers (VDNS) to resolve the DNS requests from the VMs. Each IPAM can have a virtual DNS server configured in this mode.

9. Complete the remaining fields on this page, and click **OK** to commit the changes, or click **Cancel** to clear the fields and start over.

Configuring DNS Using Scripts

DNS can be configured via the user interface or by using scripts that are available in the `opt/contrail/utils` directory. The scripts are described in [Table 40 on page 341](#).



CAUTION: Be aware of the following cautions when using scripts to configure DNS:

- DNS doesn't allow special characters in the names, other than - (dash) and . (period). Any records that include special characters in the name will be discarded by the system.
- The IPAM DNS mode and association should only be edited when there are *no* virtual machine instances in the virtual networks associated with the IPAM.

Table 40: DNS Scripts

Action	Script
Add a virtual DNS server	Script: <code>add_virtual_dns.py</code> Sample usage: <code>python add_virtual_dns.py --api_server_ip 10.204.216.21 --api_server_port 8082 --name vdns1 --domain_name default-domain --dns_domain juniper.net --dyn_updates --record_order random --ttl 1200 --next_vdns default-domain:vdns2</code>
Delete a virtual DNS server	Script: <code>del_virtual_dns_record.py</code> Sample usage: <code>python del_virtual_dns.py --api_server_ip 10.204.216.21 --api_server_port 8082 --fq_name default-domain:vdns1</code>
Add a DNS record	Script: <code>add_virtual_dns_record.py</code> Sample usage: <code>python add_virtual_dns_record.py --api_server_ip 10.204.216.21 --api_server_port 8082 --name rec1 --vdns_fqname default-domain:vdns1 --rec_name one --rec_type A --rec_class IN --rec_data 1.2.3.4 --rec_ttl 2400</code>

Table 40: DNS Scripts (*continued*)

Action	Script
Delete a DNS record	Script: <code>del_virtual_dns_record.py</code> Sample usage: <code>python del_virtual_dns_record.py --api_server_ip 10.204.216.21 --api_server_port 8082 --fq_name default-domain:vdns1:rec1</code>
Associate a virtual DNS server with an IPAM	Script: <code>associate_virtual_dns.py</code> Sample usage: <code>python associate_virtual_dns.py --api_server_ip 10.204.216.21 --api_server_port 8082 --ipam_fqname default-domain:demo:ipam1 --vdns_fqname default-domain:vdns1</code>
Disassociate a virtual DNS server with an IPAM	Script: <code>disassociate_virtual_dns.py</code> Sample usage: <code>python disassociate_virtual_dns.py --api_server_ip 10.204.216.21 --api_server_port 8082 --ipam_fqname default-domain:demo:ipam1 --vdns_fqname default-domain:vdns1</code>

Distributed Service Resource Allocation with Containerized Contrail

Starting with Contrail Release 4.0, the existing centralized Contrail discovery service is replaced with a distributed method of allocating service resources.

- [Replacement of Centralized Discovery Service in Contrail 4.0 on page 342](#)
- [New Distributed Resource Allocation Manager on page 343](#)
- [Changes in Configuration Files on page 343](#)

Replacement of Centralized Discovery Service in Contrail 4.0

In Contrail releases prior to Release 4.0, the Contrail discovery service is a centralized service resource allocation module with high availability, used primarily to automatically load-balance service resources in the system.

In the previous centralized discovery method, new service resources are registered (published) directly to the Contrail discovery module and allocated to the requester (subscriber) of the service resource, without disrupting the running state of the subscribers.

The centralized discovery method requires using a database to:

- synchronize across Contrail discovery nodes.
- maintain the list of publishers, subscribers, and the health of published services across reloads.
- provide a centralized view of the service allocation and health of the services.

This centralized discovery method resulted in unnecessary system churn when services were falsely marked as down, due to periodic health updates of services made to the database nodes, resulting in reallocation of healthy services.

Starting with Contrail 4.0, the Contrail discovery services centralized resource allocation manager has been removed. Its replacement is a distributed resource allocation list of service nodes, maintained in each module of the system.

New Distributed Resource Allocation Manager

Starting with Contrail Release 4.0, service resources are managed with a distributed allocation manager, with the following features:

- Each system module is provisioned with a list of service nodes (publishers).
- Each system module randomizes the list of service nodes and uses the resources. The randomized list is expected to be fairly load-balanced.
- When currently-used services are down, the system module detects the down immediately and reacts with no downtime by selecting another service from the list. This is distinctly different from the previous model, in which the module would need to contact the discovery service to check for available services, resulting in a finite time loss for allocation, distribution, and application of a new set of services.
- When service nodes are added or deleted, the system administrator updates the configuration file of all daemons using the service type of the service node added or deleted, sending a SIGHUP to the respective daemons.
- Each daemon randomizes the service list independently and reallocates the resources.

Deprecation of IF-MAP

In Contrail 4.0, the Interface for Metadata Access Points (IF-MAP) methodology has been deprecated. Contrail 4.0 uses CONFIGDB sections in configuration files instead of IF-MAP sections.

Changes in Configuration Files

[Table 41 on page 343](#) lists configuration files in the Contrail system that have changes to enable the distributed service resource allocation system, starting with Contrail 4.0. In general, the changes include removing (deprecating) discovery server sections and subsections, and adding parameters needed to identify service resources in all modules.

Each daemon randomizes the published service list and uses the resources. Additionally, each daemon provides a SIGHUP handler to manage the addition or deletion of publishers.

Table 41: Contrail 4.0 Changes in Configuration Files

Configuration File	Configuration Parameter	Changes
<code>contrail-vrouter-agent.conf</code>	<code>[DISCOVERY]</code>	Section deprecated
	<code>[CONTROL-NODE].servers</code>	Provisioned list of control-node [role=control] service providers in the format: ip-address:port ip-address2:port Example: 10.1.1.1:5269 10.1.1.12:5269

Table 41: Contrail 4.0 Changes in Configuration Files *(continued)*

Configuration File	Configuration Parameter	Changes
contrail-control.conf	[DNS].servers	Provisioned list of DNS [role=control] service providers in the format: ip-address:port ip-address2:port Example: 10.1.1.1:53 10.1.1.2:5
	[DEFAULT].collectors	Provisioned list of Collector [role=collector] service providers in the format: ip-address:port ip-address2:port Example: 10.1.1.1:8086 10.1.1.2:8086
	[DISCOVERY]	Section deprecated
	[DEFAULT].collectors	Provisioned list of Collector [role=collector] service providers in the format: ip-address:port ip-address2:port Example: 10.1.1.1:8086 10.1.1.2:8086
	[CONFIGDB].rabbitmq_server_list	Provisioned list of config-node [role=cfgm] service providers in the format: ip-address:port ip-address2:port Example: 10.1.1.1:5672 10.1.1.2:5672
	[CONFIGDB].rabbitmq_user	guest (default string)
	[CONFIGDB].rabbitmq_password	guest (default string)
	[CONFIGDB].config_db_server_list	Provisioned list of Config DB [role=database] service providers in the format: ip-address:port ip-address2:port Example: 10.1.1.1:9042 10.1.1.2:9042 NOTE: Docker uses 9041 as port
	[CONFIGDB].certs_store	Deprecated
	[CONFIGDB].password	Deprecated
contrail-dns.conf	[CONFIGDB].server_url	Deprecated
	[CONFIGDB].user	Deprecated
	[CONFIGDB].stale_entries_cleanup_timeout	Deprecated
	[CONFIGDB].end_of_rib_timeout	Deprecated

Table 41: Contrail 4.0 Changes in Configuration Files *(continued)*

Configuration File	Configuration Parameter	Changes
	[DISCOVERY]	Deprecated
	[DEFAULT].collectors	Provisioned list of Collector [role=collector] service providers in the format: ip-address:port ip-address2:port Example: 10.1.1.1:8086 10.1.1.2:8086
	[CONFIGDB].rabbitmq_server_list	Provisioned list of config-node [role=cfgm] service providers in the format: ip-address:port ip-address2:port Example: 10.1.1.1:5672 10.1.1.2:5672
	[CONFIGDB].rabbitmq_user	guest (default string)
	[CONFIGDB].rabbitmq_password	guest (default string)
	[CONFIGDB].config_db_server_list	Provisioned list of Config DB [role=database] service providers in the format: ip-address:port ip-address2:port Example: 10.1.1.1:9042 10.1.1.2:9042 NOTE: Dockers use 9041 as port
	[CONFIGDB].certs_store	Deprecated
	[CONFIGDB].password	Deprecated
	[CONFIGDB].server_url	Deprecated
	[CONFIGDB].user	Deprecated
	[CONFIGDB].stale_entries_cleanup_timeout	Deprecated
	[CONFIGDB].end_of_rib_timeout	Deprecated
contrail-collector.conf	[DISCOVERY]	Deprecated
	[API_SERVER].api_server_list	Provisioned list of api-servers [role=config] in the format: ip-address:port Example: 10.1.1.1:8082 10.1.1.2:8082
contrail-alarm-gen.conf	[DISCOVERY]	Deprecated

Table 41: Contrail 4.0 Changes in Configuration Files *(continued)*

Configuration File	Configuration Parameter	Changes
	[DEFAULTS].collectors	Provisioned list of Collector [role=collector] service providers in the format: ip-address:port ip-address2:port Example: 10.1.1.1:8086 10.1.1.2:8086
	[API_SERVER].api_server_list	Provisioned list of api-servers [role=config] in the format: ip-address:port Example: 10.1.1.1:8082 10.1.1.2:8082
	[REDIS].redis_uve_list	Provisioned list of redis instances [role=collector] Example: 192.168.0.29:6379 192.168.0.30:6379
contrail-analytics-api.conf	[DISCOVERY]	Section deprecated
	[DEFAULTS].collectors	Provisioned list of collector [role=collector] service providers in the format: ip-address:port ip-address2:port Example: 10.1.1.1:8086 10.1.1.2:8086
	[REDIS].redis_uve_list	Provisioned list of redis instances [role=collector] Example: 192.168.0.29:6379 192.168.0.30:6379
contrail-api.conf	[DISCOVERY]	Section deprecated
	[DEFAULTS].collectors	Provisioned list of collector [role=collector] service providers in the format: ip-address:port ip-address2:port Example: 10.1.1.1:8086 10.1.1.2:8086
contrail-schema.conf	[DISCOVERY]	Section deprecated
	[DEFAULTS].collectors	Provisioned list of Collector [role=collector] service providers in ip-address:port ip-address2:port Example: 10.1.1.1:8086 10.1.1.2:8086
contrail-svc-monitor.conf	[DISCOVERY]	Section deprecated

Table 41: Contrail 4.0 Changes in Configuration Files *(continued)*

Configuration File	Configuration Parameter	Changes
	[DEFAULTS].collectors	Provisioned list of Collector [role=collector] service providers in the format: ip-address:port ip-address2:port Example: 10.1.1.1:8086 10.1.1.2:8086
contrail-device-manager.conf	[DISCOVERY]	Section deprecated
	[DEFAULTS].collectors	Provisioned list of Collector [role=collector] service providers in ip-address:port ip-address2:port format Example: 10.1.1.1:8086 10.1.1.2:8086
contrail-analytics-nodemgr.conf	[DISCOVERY]	Section deprecated
	[COLLECTOR].server_list	Provisioned list of Collector [role=collector] service providers in the format: ip-address:port ip-address2:port Example: 10.1.1.1:8086 10.1.1.2:8086
contrail-config-nodemgr.conf	[DISCOVERY]	Section deprecated
	[COLLECTOR].server_list	Provisioned list of Collector [role=collector] service providers in the format: ip-address:port ip-address2:port Example: 10.1.1.1:8086 10.1.1.2:8086
contrail-control-nodemgr.conf	[DISCOVERY]	Section deprecated
	[COLLECTOR].server_list	Provisioned list of Collector [role=collector] service providers in ip-address:port ip-address2:port Example: 10.1.1.1:8086 10.1.1.2:8086
contrail-database-nodemgr.conf	[DISCOVERY]	Section deprecated
	[COLLECTOR].server_list	Provisioned list of Collector [role=collector] service providers in the format: ip-address:port ip-address2:port Example: 10.1.1.1:8086 10.1.1.2:8086
contrail-vrouter-nodemgr.conf	[DISCOVERY]	Section deprecated

Table 41: Contrail 4.0 Changes in Configuration Files *(continued)*

Configuration File	Configuration Parameter	Changes
	<code>[COLLECTOR].server_list</code>	Provisioned list of Collector [role=collector] service providers in the format: ip-address:port ip-address2:port Example: 10.1.1.1:8086 10.1.1.2:8086
<code>contrail-query-engine.conf</code>	<code>[DISCOVERY]</code>	Section deprecated
	<code>[COLLECTOR].server_list</code>	Provisioned list of Collector [role=collector] service providers in the format: ip-address:port ip-address2:port Example: 10.1.1.1:8086 10.1.1.2:8086
<code>contrail-snmp-collector.conf</code>	<code>[DISCOVERY]</code>	Section deprecated
	<code>[DEFAULTS].collectors</code>	Provisioned list of Collector [role=collector] service providers in the format: ip-address:port ip-address2:port Example: 10.1.1.1:8086 10.1.1.2:8086
	<code>[API_SERVER].api_server_list</code>	Provisioned list of api-servers [role=config] in the format: ip-address:port Example: 10.1.1.1:8082 10.1.1.2:8082
<code>contrail-topology.conf</code>	<code>[DISCOVERY]</code>	Section deprecated
	<code>[DEFAULTS].collectors</code>	Provisioned list of Collector [role=collector] service providers in the format: ip-address:port ip-address2:port Example: 10.1.1.1:8086 10.1.1.2:8086
	<code>[API_SERVER].api_server_list</code>	Provisioned list of api-servers [role=config] in ip-address:port Example: 10.1.1.1:8082 10.1.1.2:8082
Contrail Web UI		
<code>config.global.js</code>	<code>config.discovery.server</code>	Discovery subsection deprecated
	<code>config.discovery.port</code>	Discovery subsection deprecated

Table 41: Contrail 4.0 Changes in Configuration Files (*continued*)

Configuration File	Configuration Parameter	Changes
	<code>config.cnfg.server_ip</code>	Provisioned list of Config [role=cfgm] service providers as list of ip-address Example: ['10.1.1.1 10.1.1.2']
	<code>config.cnfg.server_port</code>	Server port as a string Example: '8082'
	<code>config.analytics.server_ip</code>	Provisioned list of Collector [role=collector] service providers as a list of ip-address Example: ['10.1.1.1 10.1.1.2']
	<code>config.analytics.server_port</code>	Server port as a string Example: '8081'
	<code>config.dns.server_ip</code>	Provisioned list of Controller [role=control] service providers as a list of ip-address Example: ['10.1.1.1 10.1.1.2']
	<code>config.dns.server_port</code>	Server port as a string Example: '8092'

Support for Multicast

This section describes how the Contrail Controller supports broadcast and multicast.

- [Subnet Broadcast on page 350](#)
- [All-Broadcast/Limited-Broadcast and Link-Local Multicast on page 350](#)
- [Host Broadcast on page 351](#)

Subnet Broadcast

Multiple subnets can be attached to a virtual network when it is spawned. Each of the subnets has one subnet broadcast route installed in the unicast routing table assigned to that virtual network. The recipient list for the subnet broadcast route includes all of the virtual machines that belong to that subnet. Packets originating from any VM in that subnet are replicated to all members of the recipient list, except the originator. Because the next hop is the list of recipients, it is called a composite next hop.

If there is no virtual machine spawned under a subnet, the subnet routing entry discards the packets received. If all of the virtual machines in a subnet are turned off, the routing entry points to discard. If the IPAM is deleted, the subnet route corresponding to that IPAM is deleted. If the virtual network is turned off, all of the subnet routes associated with the virtual network are removed.

Subnet Broadcast Example

The following configuration is made:

Virtual network name – **vn1**
Unicast routing instance – **vn1.uc.inet**
Subnets (IPAM) allocated – **1.1.1.0/24; 2.2.0.0/16; 3.3.0.0/16**
Virtual machines spawned – **vm1 (1.1.1.253); vm2 (1.1.1.252); vm3 (1.1.1.251); vm4 (3.3.1.253)**

The following subnet route additions are made to the routing instance **vn1.uc.inet.0**:

1.1.1.255 -> forward to NH1 (composite next hop)
2.2.255.255 -> DROP
3.3.255.255 -> forward to NH2

The following entries are made to the next-hop table:

NH1 – **1.1.1.253; 1.1.1.252; 1.1.1.251**
NH2 – **3.3.1.253**

If traffic originates for **1.1.1.255** from **vm1 (1.1.1.253)**, it will be forwarded to **vm2 (1.1.1.252)** and **vm3 (1.1.1.251)**. The originator **vm1 (1.1.1.253)** will not receive the traffic even though it is listed as a recipient in the next hop.

All-Broadcast/Limited-Broadcast and Link-Local Multicast

The address group **255.255.255.255** is used with all-broadcast (limited-broadcast) and multicast traffic. The route is installed in the multicast routing instance. The source address is recorded as ANY, so the route is **ANY/255.255.255.255 (*G)**. It is unique per routing instance, and is associated with its corresponding virtual network. When a virtual network is spawned, it usually contains multiple subnets, in which virtual machines are added. All of the virtual machines, regardless of their subnets, are part of the recipient list for **ANY/255.255.255.255**. The replication is sent to every recipient except the originator.

Link-local multicast also uses the all-broadcast method for replication. The route is deleted when all virtual machines in this virtual network are turned off or the virtual network itself is deleted.

All-Broadcast Example

The following configuration is made:

Virtual network name – **vn1**

Unicast routing instance – **vn1.uc.inet**

Subnets (IPAM) allocated – **1.1.1.0/24; 2.2.0.0/16; 3.3.0.0/16**

Virtual machines spawned – **vm1 (1.1.1.253); vm2 (1.1.1.252); vm3 (1.1.1.251); vm4 (3.3.1.253)**

The following subnet route addition is made to the routing instance **vn1.uc.inet.0**:

255.255.255.255/* -> NH1

The following entries are made to the next-hop table:

NH1 – 1.1.1.253; 1.1.1.252; 1.1.1.251; 3.3.1.253

If traffic originates for **1.1.1.255** from **vm1 (1.1.1.253)**, the traffic is forwarded to **vm2 (1.1.1.252)**, **vm3 (1.1.1.251)**, and **vm4 (3.3.1.253)**. The originator **vm1 (1.1.1.253)** will not receive the traffic even though it is listed as a recipient in the next hop.

Host Broadcast

The host broadcast route is present in the host routing instance so that the host operating system can send a subnet broadcast/all-broadcast (limited-broadcast). This type of broadcast is sent to the fabric by means of a **vhost** interface. Additionally, any subnet broadcast/all-broadcast received from the fabric will be handed over to the host operating system.

Using Static Routes with Services

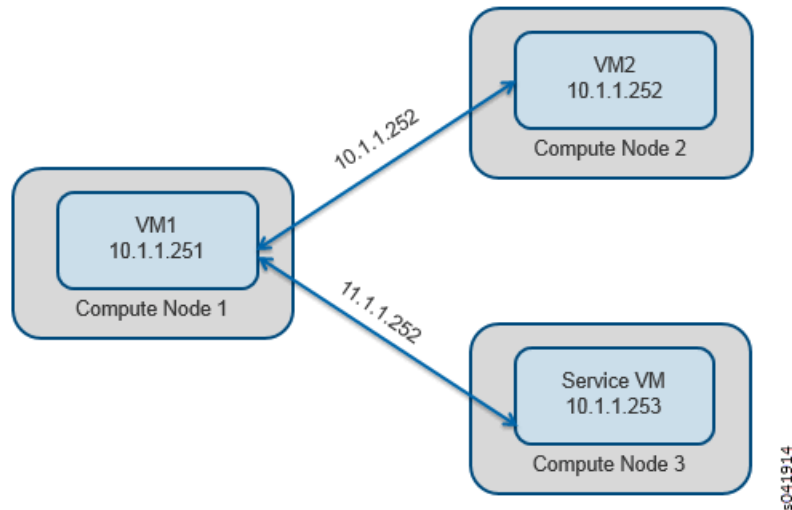
- [Static Routes for Service Instances on page 351](#)
- [Configuring Static Routes on a Service Instance on page 352](#)
- [Configuring Static Routes on Service Instance Interfaces on page 353](#)
- [Configuring Static Routes as Host Routes on page 355](#)

Static Routes for Service Instances

Static routes can be configured in a virtual network to direct traffic to a service virtual machine.

The following figure shows a virtual network with subnet 10.1.1.0/24. All of the traffic from a virtual machine that is directed to subnet 11.1.1.0/24 can be configured to be routed by

means of a service machine, by using the static route 11.1.1.252 configured on the service virtual machine interface.



Configuring Static Routes on a Service Instance

To configure static routes on a service instance, first enable the static route option in the service template to be used for the service instance.

To enable the static route option in a service template:

1. Go to **Configure > Services > Service Templates** and click **Create**.
2. At **Add Service Template**, complete the fields for **Name**, **Service Mode**, and **Image Name**.
3. Select the **Interface Types** to use for the template, then for each interface type that might have a static route configured, click the check box under the **Static Routes** column to enable the static route option for that interface.

The following figure shows a service template in which the left and right interfaces of service instances have the static routes option enabled. Now a user can configure

a static route on a corresponding interface on a service instance that is based on the service template shown.

Add Service Template

Name:

Service Mode:

Image Name:

Interface Types	Shared IP	Static Routes	+	-
Management	<input type="checkbox"/>	<input type="checkbox"/>	+	-
Left	<input type="checkbox"/>	<input checked="" type="checkbox"/>	+	-
Right	<input type="checkbox"/>	<input checked="" type="checkbox"/>	+	-

▶ [Advanced options](#)

Cancel Save

5041915

Configuring Static Routes on Service Instance Interfaces

To configure static routes on a service instance interface:

1. Go to **Configure > Services > Service Instances** and click **Create**.
2. At **Create Service Instances**, complete the fields for **Instance Name** and **Services Template**.
3. Select the virtual network for each of the interfaces
4. Click the **Static Routes** dropdown menu under each interface field for which the static routes option is enabled to open the **Static Routes** menu and configure the static routes in the fields provided.



NOTE: If the **Auto Configured** option is selected, traffic destined to the static route subnet is load balanced across service instances.

The following figure shows a configuration to apply a service instance between VN1 (10.1.1.0/24) and VN2 (11.1.1.0/24). The left interface of the service instance is configured with VN1 and the right interface is configured to be VN2 (11.1.1.0/24). The static route 11.1.1.0/24 is configured on the left interface, so that all traffic from VN1 that is destined to VN2 reaches the left interface of the service instance.

The screenshot shows the 'Create Service Instances' dialog box. The 'Instance Name' is 'nat'. The 'Services Template' is 'nat - [in-network (management, left, right)]'. Under 'Interface 1', 'Management' is selected with 'Auto Configured'. Under 'Interface 2', 'Left' is selected with 'vn1'. A 'Static Routes' section for Interface 2 shows a table with one entry: Prefix '11.1.1.0/24', Next hop 'Interface 2'. Under 'Interface 3', 'Right' is selected with 'vn2'. A 'Static Routes' section for Interface 3 is visible but empty. The bottom right corner shows a timestamp 's04:19:16' and 'Cancel'/'Save' buttons.

Interface	Interface Name	Next Hop	Prefix
Interface 2 (Left)	vn1	Interface 2	11.1.1.0/24
Interface 3 (Right)	vn2		

The following figure shows static route 10.1.1.0/24 configured on the right interface, so that all traffic from VN2 that is destined to VN1 reaches the right interface of the service virtual machine.

The screenshot shows the 'Create Service Instances' dialog box. The 'Instance Name' is 'nat'. The 'Services Template' is 'nat - [in-network (management, left, right)]'. Under 'Interface 2', 'Left' is selected with 'vn1'. A 'Static Routes' section for Interface 2 shows a table with one entry: Prefix '11.1.1.0/24', Next hop 'Interface 2'. Under 'Interface 3', 'Right' is selected with 'vn2'. A 'Static Routes' section for Interface 3 shows a table with one entry: Prefix '10.1.1.0/24', Next hop 'Interface 3'. The bottom right corner shows a timestamp 's04:19:17' and 'Cancel'/'Save' buttons.

Interface	Interface Name	Next Hop	Prefix
Interface 2 (Left)	vn1	Interface 2	11.1.1.0/24
Interface 3 (Right)	vn2	Interface 3	10.1.1.0/24

When the static routes are configured for both the left and the right interfaces, all inter-virtual network traffic is forwarded through the service instance.

Configuring Static Routes as Host Routes

You can also use static routes for host routes for a virtual machine, by using the classless static routes option in the DHCP server response that is sent to the virtual machine.

The routes to be sent in the DHCP response to the virtual machine can be configured for each virtual network as it is created.

To configure static routes as host routes:

1. Go to **Configure > Network > Networks** and click **Create**.
2. At **Create Network**, click the **Host Routes** option and add the host routes to be sent to the virtual machines.

An example is shown in the following figure.

The screenshot shows the 'Create Network' dialog box. The 'Address Management' section is expanded, showing a table with columns 'IPAM', 'IP Block', and 'Gateway'. The table contains one row with 'ipam1', '1.2.3.0/24', and '1.2.3.254'. Below this, the 'Host Routes' section is expanded, showing a table with columns 'IPAM' and 'Route Prefix'. The table contains two rows: 'ipam1' with '1.1.1.0/24' and 'ipam1' with '2.2.2.0/24'. The '2.2.2.0/24' row is highlighted. The dialog box has 'Cancel' and 'Save' buttons at the bottom right.

IPAM	IP Block	Gateway
ipam1	1.2.3.0/24	1.2.3.254

IPAM	Route Prefix
ipam1	1.1.1.0/24
ipam1	2.2.2.0/24

Configuring Metadata Service

OpenStack enables virtual machines to access metadata by sending an HTTP request to the link-local address 169.254.169.254. The metadata request from the virtual machine is proxied to Nova with additional HTTP header fields that Nova uses to identify the source instance, then responds with appropriate metadata.

In Contrail, the vRouter acts as the proxy, by trapping the metadata requests, adding the necessary header fields, and sending the requests to the Nova API server.

The metadata service is configured by setting the **linklocal-services** property on the **global-vrouter-config** object.

Use the following elements to configure the **linklocal-services** element for metadata service:

- **linklocal-service-name** = metadata
- **linklocal-service-ip** = 169.254.169.254
- **linklocal-service-port** = 80
- **ip-fabric-service-ip** = *[server-ip-address]*
- **ip-fabric-service-port** = *[server-port]*

The **linklocal-services** properties can be set from the Contrail UI (**Configure > Infrastructure > Link Local Services**) or by using the following command:

```
python /opt/contrail/utils/provision_linklocal.py --admin_user <user> --admin_password  
<passwd> --linklocal_service_name metadata --linklocal_service_ip 169.254.169.254  
--linklocal_service_port 80 --ipfabric_service_ip --ipfabric_service_port 8775
```

CHAPTER 15

Configuring Service Chaining

- [Service Chaining on page 357](#)
- [Service Chaining MX Series Configuration on page 361](#)
- [ECMP Load Balancing in the Service Chain on page 362](#)
- [Customized Hash Field Selection for ECMP Load Balancing on page 363](#)
- [Service Chain Version 2 with Port Tuple on page 367](#)
- [Using the Contrail Heat Template on page 371](#)
- [Service Chain Route Reorigination on page 374](#)
- [Health Check Object on page 392](#)

Service Chaining

Contrail Controller supports chaining of various Layer 2 through Layer 7 services such as firewall, NAT, IDP, and so on.

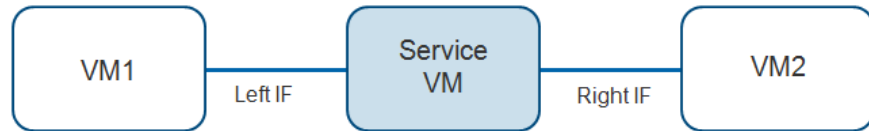
- [Service Chaining Basics on page 357](#)
- [Service Chaining Configuration Elements on page 359](#)

Service Chaining Basics

Services are offered by instantiating service virtual machines to dynamically apply single or multiple services to virtual machine (VM) traffic. It is also possible to chain physical appliance-based services.

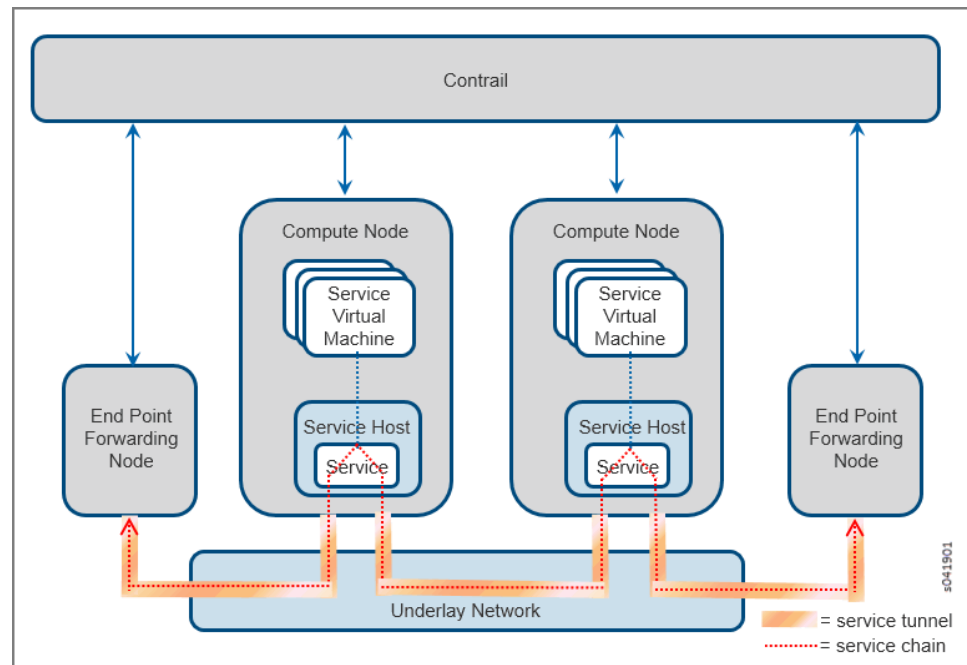
[Figure 123 on page 358](#) shows the basic service chain schema, with a single service. The service VM spawns the service, using the convention of left interface (left IF) and right interface (right IF). Multiple services can also be chained together.

Figure 123: Service Chaining



When you create a service chain, the Contrail software creates tunnels across the underlay network that span through all services in the chain. [Figure 124 on page 358](#) shows two end points and two compute nodes, each with one service instance and traffic going to and from one end point to the other.

Figure 124: Contrail Service Chain



The following are the modes of services that can be configured.

Transparent or bridge mode

Used for services that do not modify the packet. Also known as bump-in-the-wire or Layer 2 mode. Examples include Layer 2 firewall, IDP, and so on.

In-network or routed mode

Provides a gateway service where packets are routed between the service instance interfaces. Examples include NAT, Layer 3 firewall, load balancer, HTTP proxy, and so on.

In-network-nat mode

Similar to in-network mode, however, return traffic does not need to be routed to the source network. In-network-nat mode is particularly useful for NAT service.

Service Chaining Configuration Elements

Service chaining requires the following configuration elements in the solution:

- Service template
- Service instance
- Service policy

Service Template

Service templates are always configured in the scope of a domain, and the templates can be used on all projects within a domain. A template can be used to launch multiple service instances in different projects within a domain.

The following are the parameters to be configured for a service template:

- Service template name
- Domain name
- Service mode
 - Transparent
 - In-Network
 - In-Network NAT
- Image name (for virtual service)
 - If the service is a virtual service, then the name of the image to be used must be included in the service template. In an OpenStack setup, the image must be added to the setup by using Glance.
- Interface list
 - Ordered list of interfaces---this determines the order in which Interfaces will be created on the service instance.
 - Most service templates will have management, left, and right interfaces. For service instances requiring more interfaces, “other” interfaces can be added to the interface list.

- Shared IP attribute, per interface
- Static routes enabled attribute, per interface
- Advanced options
 - Service scaling— use this attribute to enable a service instance to have more than one instance of the service instance virtual machine.
 - Flavor—assign an OpenStack flavor to be used while launching the service instance. Flavors are defined in OpenStack Nova with attributes such as assignments of CPU cores, memory, and disk space.

Service Instance

A service instance is always maintained within the scope of a project. A service instance is launched using a specified service template from the domain to which the project belongs.

The following are the parameters to be configured for a service instance:

- Service instance name
- Project name
- Service template name
- Number of virtual machines that will be spawned
 - Enable service scaling in the service template for multiple virtual machines
- Ordered virtual network list
 - Interfaces listed in the order specified in the service template
 - Identify virtual network for each interface
 - Assign static routes for virtual networks that have static route enabled in the service template for their interface
 - Traffic that matches an assigned static route is directed to the service instance on the interface created for the corresponding virtual network

Service Policy

The following are the parameters to be configured for a service policy:

- Policy name
- Source network name
- Destination network name
- Other policy match conditions, for example direction and source and destination ports
- Policy configured in “routed/in-network” or “bridged/” mode
- An action type called **apply_service** is used:

Example: 'apply_service': [DomainName:ProjectName:ServiceInstanceName]

**Related
Documentation**

- [Example: Creating an In-Network or In-Network-NAT Service Chain on page 397](#)
- [Example: Creating a Service Chain With the CLI on page 409](#)
- [ECMP Load Balancing in the Service Chain on page 362](#)

Service Chaining MX Series Configuration

This topic shows how to extend service chaining to the MX Series routers.

To configure service chaining for MX Series routers, extend the virtual networks to the MX Series router and program routes so that traffic generated from a host connected to the router can be routed through the service.

1. The following configuration snippet for an MX Series router has a left virtual network called **enterprise** and a right virtual network called **public**. The configuration creates two routing instances with loopback interfaces and route targets.

```
routing-instances {
  enterprise {
    instance-type vrf;
    interface lo0.1;
    vrf-target target:100:20000;
  }
  public {
    instance-type vrf;
    interface lo0.2;
    vrf-target target:100:10000;
  }
  routing-options {
    static {
      route 0.0.0.0/0 next-hop 10.84.20.1
    }
  }
  interface xe-0/0/0.0;
}
```

2. The following configuration snippet shows the configuration for the loopback interfaces.

```
interfaces {
  lo0 {
    unit 1 {
      family inet {
        address 2.1.1.100/32;
      }
    }
    unit 2 {
      family inet {
        address 200.1.1.1/32;
      }
    }
  }
}
```

```

    }
}

```

- The following configuration snippet shows the configuration to enable BGP. The **neighbor 10.84.20.39** and **neighbor 10.84.20.40** are control nodes.

```

protocols {
  bgp {
    group demo_contrail {
      type internal;
      description "To Contrail Control Nodes & other MX";
      local-address 10.84.20.252;
      keep all;
      family inet-vpn {
        unicast;
      }
      neighbor 10.84.20.39;
      neighbor 10.84.20.40;
    }
  }
}

```

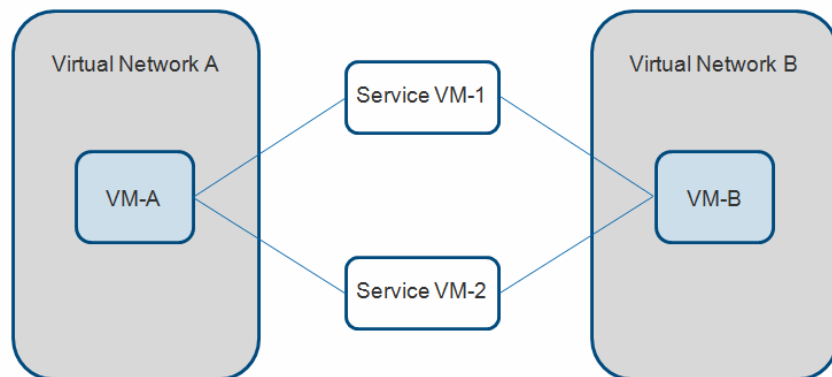
- The final step is to add **target:100:10000** to the public virtual network and **target:100:20000** to the enterprise virtual network, using the Contrail Juniper Networks interface.

A full MX Series router configuration for Contrail can be seen in [“Sample Network Configuration for Devices for Simple Tiered Web Application”](#) on page 327.

ECMP Load Balancing in the Service Chain

Traffic flowing through a service chain can be load-balanced by distributing traffic streams to multiple service virtual machines (VMs) that are running identical applications. This is illustrated in [Figure 125 on page 362](#), where the traffic streams between VM-A and VM-B are distributed between Service VM-1 and Service VM-2. If Service VM-1 goes down, then all streams that are dependent on Service VM-1 will be moved to Service VM-2.

Figure 125: Load Balancing a Service Chain



s041830

The following are the major features of load balancing in the service chain:

- Load balancing can be configured at every level of the service chain.
- Load balancing is supported in routed and bridged service chain modes.
- Load balancing can be used to achieve high availability—if a service VM goes down, the traffic passing through that service VM can be distributed through another service VM.
- A load balanced traffic stream always follows the same path through the chain of service VM.

**Related
Documentation**

- [Service Chaining on page 357](#)
- [Example: Creating a Service Chain With the CLI on page 409](#)
- [Customized Hash Field Selection for ECMP Load Balancing on page 363](#)

Customized Hash Field Selection for ECMP Load Balancing

Overview: Custom Hash Feature

Starting with Contrail Release 3.0, it is possible to configure the set of fields used to hash upon during equal-cost multipath (ECMP) load balancing.

Earlier versions of Contrail had this set of fields fixed to the standard 5-tuple set of: source L3 address, destination L3 address, L4 protocol, L4 SourcePort, and L4 DestinationPort.

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.

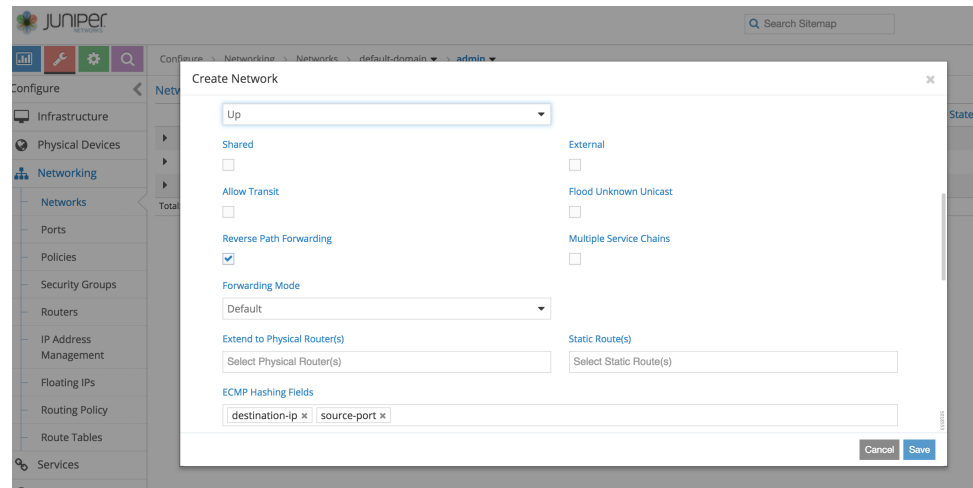
The custom hash configuration can be applied in the following ways:

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

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

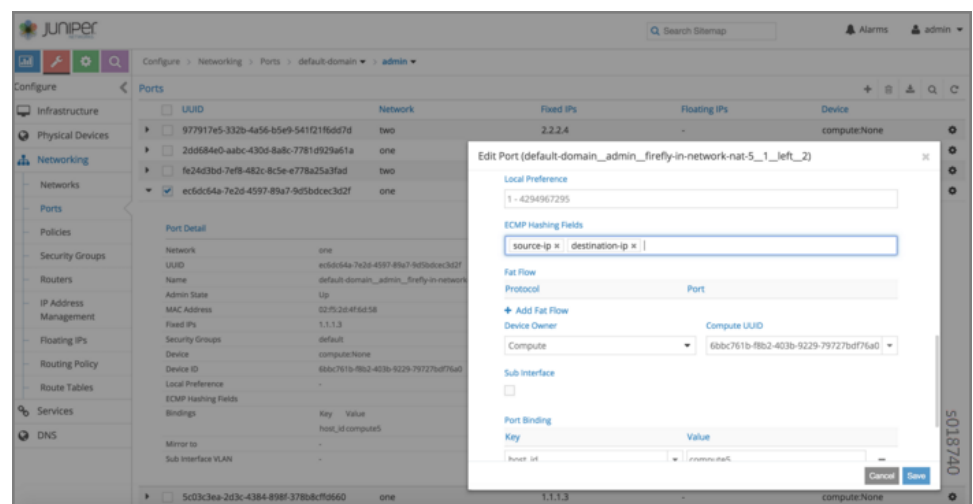
Custom hash is useful whenever packets originating from a particular source and addressed to a particular destination must go through the same set of service instances during transit. This might be required if source, destination, or transit nodes maintain a certain state based on the flow, and the state 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 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.



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 (VMI) of the service instance. Then, each service chain route originating from that VMI 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.



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 UI Ports section under Networking, select the left port (VMI) 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.

Sample Flows

This section provides sample flows with and without ECMP custom hash field selection.

Sample Traffic Flow Path Without Custom ECMP Hash Fields

The following is an example of a traffic flow path without using a customized ECMP hash fields selection configuration. The flow is configured with standard 5-tuple flow fields.

```
tcpdump -i eth0 'port 1023 and tcp[tcpflags] & (tcp-syn) != 0 and tcp[tcpflags]
& (tcp-ack) == 0'
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
14:55:10.115122 IP 2.2.2.5.18337 > 2.2.2.100.1023: Flags [S], seq 2276852196, win
29200, options [mss 1398,sackOK,TS val 25208882 ecr 0,nop,wscale 7], length 0
14:55:10.132753 IP 2.2.2.4.21193 > 2.2.2.100.1023: Flags [S], seq 4161487314, win
29200, options [mss 1398,sackOK,TS val 25208886 ecr 0,nop,wscale 7], length 0
14:55:10.152053 IP 2.2.2.5.24230 > 2.2.2.100.1023: Flags [S], seq 2466454857, win
29200, options [mss 1398,sackOK,TS val 25208892 ecr 0,nop,wscale 7], length 0
14:55:11.146029 IP 2.2.2.5.24230 > 2.2.2.100.1023: Flags [S], seq 2466454857, win
29200, options [mss 1398,sackOK,TS val 25209142 ecr 0,nop,wscale 7], length 0
14:55:13.147616 IP 2.2.2.5.24230 > 2.2.2.100.1023: Flags [S], seq 2466454857, win
29200, options [mss 1398,sackOK,TS val 25209643 ecr 0,nop,wscale 7], length 0
14:55:13.164367 IP 2.2.2.3.25582 > 2.2.2.100.1023: Flags [S], seq 2259034580, win
29200, options [mss 1398,sackOK,TS val 25209644 ecr 0,nop,wscale 7], length 0
14:55:13.179939 IP 2.2.2.5.24895 > 2.2.2.100.1023: Flags [S], seq 2174031724, win
29200, options [mss 1398,sackOK,TS val 25209648 ecr 0,nop,wscale 7], length 0
14:55:14.168282 IP 2.2.2.5.24895 > 2.2.2.100.1023: Flags [S], seq 2174031724, win
29200, options [mss 1398,sackOK,TS val 25209898 ecr 0,nop,wscale 7], length 0
14:55:16.172384 IP 2.2.2.5.24895 > 2.2.2.100.1023: Flags [S], seq 2174031724, win
29200, options [mss 1398,sackOK,TS val 25210399 ecr 0,nop,wscale 7], length 0
14:55:16.189864 IP 2.2.2.5.22952 > 2.2.2.100.1023: Flags [S], seq 3099816842, win
29200, options [mss 1398,sackOK,TS val 25210401 ecr 0,nop,wscale 7], length 0
14:55:16.205142 IP 2.2.2.4.16487 > 2.2.2.100.1023: Flags [S], seq 3961114202, win
29200, options [mss 1398,sackOK,TS val 25210405 ecr 0,nop,wscale 7], length 0
14:55:17.196763 IP 2.2.2.4.16487 > 2.2.2.100.1023: Flags [S], seq 3961114202, win
29200, options [mss 1398,sackOK,TS val 25210655 ecr 0,nop,wscale 7], length 0
14:55:19.200623 IP 2.2.2.4.16487 > 2.2.2.100.1023: Flags [S], seq 3961114202, win
29200, options [mss 1398,sackOK,TS val 25211156 ecr 0,nop,wscale 7], length 0
14:55:19.215809 IP 2.2.2.3.18914 > 2.2.2.100.1023: Flags [S], seq 3157557440, win
29200, options [mss 1398,sackOK,TS val 25211158 ecr 0,nop,wscale 7], length 0
14:55:19.228405 IP 2.2.2.7.15569 > 2.2.2.100.1023: Flags [S], seq 3850648420, win
29200, options [mss 1398,sackOK,TS val 25211161 ecr 0,nop,wscale 7], length 0
14:55:20.223482 IP 2.2.2.7.15569 > 2.2.2.100.1023: Flags [S], seq 3850648420, win
29200, options [mss 1398,sackOK,TS val 25211412 ecr 0,nop,wscale 7], length 0
14:55:22.232068 IP 2.2.2.7.15569 > 2.2.2.100.1023: Flags [S], seq 3850648420, win
29200, options [mss 1398,sackOK,TS val 25211913 ecr 0,nop,wscale 7], length 0
14:55:22.247325 IP 2.2.2.4.28388 > 2.2.2.100.1023: Flags [S], seq 3609240658, win
29200, options [mss 1398,sackOK,TS val 25211915 ecr 0,nop,wscale 7], length 0
```

Sample Traffic Flow Path With Custom ECMP Hash Fields

The following is an example of a traffic flow path using a customized ECMP hash fields selection configuration, for **source-ip** and **destination-ip** only.

```
tcpdump -i eth0 'port 1023 and tcp[tcpflags] & (tcp-syn) != 0 and tcp[tcpflags]
& (tcp-ack) == 0'
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
```

```

listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
15:57:18.680853 IP 2.2.2.4.21718 > 2.2.2.100.1023: Flags [S], seq 2052086108, win
 29200, options [mss 1398,sackOK,TS val 26141024 ecr 0,nop,wscale 7], length 0
15:57:18.696114 IP 2.2.2.4.13585 > 2.2.2.100.1023: Flags [S], seq 2039627277, win
 29200, options [mss 1398,sackOK,TS val 26141028 ecr 0,nop,wscale 7], length 0
15:57:18.714846 IP 2.2.2.4.16414 > 2.2.2.100.1023: Flags [S], seq 3252526560, win
 29200, options [mss 1398,sackOK,TS val 26141033 ecr 0,nop,wscale 7], length 0
15:57:18.731281 IP 2.2.2.4.32499 > 2.2.2.100.1023: Flags [S], seq 1389133175, win
 29200, options [mss 1398,sackOK,TS val 26141037 ecr 0,nop,wscale 7], length 0
15:57:18.747051 IP 2.2.2.4.6081 > 2.2.2.100.1023: Flags [S], seq 427936299, win
 29200, options [mss 1398,sackOK,TS val 26141041 ecr 0,nop,wscale 7], length 0
15:57:19.740204 IP 2.2.2.4.6081 > 2.2.2.100.1023: Flags [S], seq 427936299, win
 29200, options [mss 1398,sackOK,TS val 26141291 ecr 0,nop,wscale 7], length 0
15:57:21.743951 IP 2.2.2.4.6081 > 2.2.2.100.1023: Flags [S], seq 427936299, win
 29200, options [mss 1398,sackOK,TS val 26141792 ecr 0,nop,wscale 7], length 0
15:57:21.758532 IP 2.2.2.4.13800 > 2.2.2.100.1023: Flags [S], seq 3020971712, win
 29200, options [mss 1398,sackOK,TS val 26141794 ecr 0,nop,wscale 7], length 0
15:57:21.772646 IP 2.2.2.4.23894 > 2.2.2.100.1023: Flags [S], seq 3373734307, win
 29200, options [mss 1398,sackOK,TS val 26141797 ecr 0,nop,wscale 7], length 0
15:57:22.764469 IP 2.2.2.4.23894 > 2.2.2.100.1023: Flags [S], seq 3373734307, win
 29200, options [mss 1398,sackOK,TS val 26142047 ecr 0,nop,wscale 7], length 0
15:57:24.768511 IP 2.2.2.4.23894 > 2.2.2.100.1023: Flags [S], seq 3373734307, win
 29200, options [mss 1398,sackOK,TS val 26142548 ecr 0,nop,wscale 7], length 0
15:57:24.784119 IP 2.2.2.4.21858 > 2.2.2.100.1023: Flags [S], seq 2212369297, win
 29200, options [mss 1398,sackOK,TS val 26142550 ecr 0,nop,wscale 7], length 0
15:57:24.797149 IP 2.2.2.4.29440 > 2.2.2.100.1023: Flags [S], seq 2007897735, win
 29200, options [mss 1398,sackOK,TS val 26142554 ecr 0,nop,wscale 7], length 0
15:57:25.792816 IP 2.2.2.4.29440 > 2.2.2.100.1023: Flags [S], seq 2007897735, win
 29200, options [mss 1398,sackOK,TS val 26142804 ecr 0,nop,wscale 7], length 0
15:57:27.797538 IP 2.2.2.4.29440 > 2.2.2.100.1023: Flags [S], seq 2007897735, win
 29200, options [mss 1398,sackOK,TS val 26143305 ecr 0,nop,wscale 7], length 0
15:57:27.814002 IP 2.2.2.4.23452 > 2.2.2.100.1023: Flags [S], seq 1659332655, win
 29200, options [mss 1398,sackOK,TS val 26143307 ecr 0,nop,wscale 7], length 0

```

Service Chain Version 2 with Port Tuple

Starting with Contrail 3.0, the user can create a port-tuple object for binding service instances to ports.

- [Overview of Port Tuple on page 367](#)
- [Service Chain Version 2 Sample Workflow on page 368](#)
- [Service Chain with Health Check on page 370](#)

Overview of Port Tuple

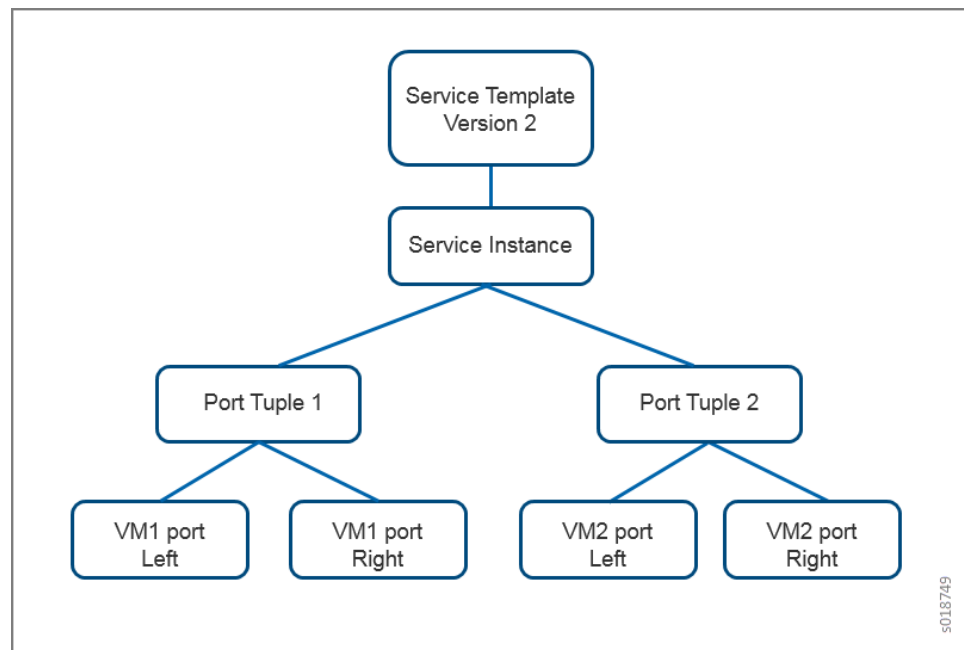
In previous versions of Contrail, when a service instance is created for a virtual machine (VM)-based service, the service monitor creates one or more VM objects and creates a port for each VM object. Each VM object is a placeholder for binding a service instance to a port. The VM object also acts as a placeholder for the instance ID when using equal-cost multipath (ECMP).

Using the VM object as a placeholder doesn't add value beyond binding information between the service instance object and the port objects. By using a port-tuple object, the service instance can be linked directly to the port objects, eliminating the need to

create a VM object. This simplifies the implementation of service instance objects, and also allows integration with Heat templates.

With a port-tuple object, the user can create ports and pass the port information when creating a service instance. The ports can be created as part of a VM launch from Nova or without using a VM launch. The ports are linked to a port-tuple object that is a child of a service instance. This functionality can also be leveraged in Heat stacks. See [Figure 126 on page 368](#).

Figure 126: Port Tuple Overview



Service Chain Version 2 Sample Workflow

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

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

1. Create a service template. Select 2 in the Version field.
2. Create a service instance for the service template just created.
3. Create a port-tuple object.
4. Create ports, using Nova VM launch or without a VM launch.

5. Label each port as **left**, **right**, **mgmt**, and so on, and add the ports to the port-tuple object.

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

6. Link the port tuple to a service instance.

7. Launch the service instance. This creates the necessary objects in the Contrail database.



NOTE: Port-tuple is *not* supported on transparent service instance, whether active/active, active/standby, or scale-out.

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

Equal-cost multipath (ECMP) can be used to distribute traffic across VMs. 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 Active-Standby Mode with Allowed Address Pair

To support **active-standby** mode, you must configure an allowed address pair on the interfaces. The **active-standby** is used as the high availability mode in the allowed address pair. The allowed address pair is configured as part of the service instance for a particular VM port label. For example, if the allowed address pair is configured in a service instance for the port with the label **left**, then all of the port-tuple VM ports with the label **left** will use the allowed address pair high availability mode.

Allowed Address Pair

An allowed address pair extension is an OpenStack feature supported by Contrail.

By default, there is no way to specify additional MAC/IP address pairs that are allowed to pass through a port in Neutron, because ports are locked down to their MAC address and the fixed IPs associated with their port for anti-spoofing reasons. This locking can sometimes prevent protocols such as VRRP from providing a high availability failover strategy. Using the allowed address pair extension enables additional IP/MAC pairs to be allowed through ports in Neutron.

In Contrail, you can configure allowed address pairs in the service instance configuration, using **Configure > Services > Service Instances > Allowed Address Pair**, see

[Figure 127 on page 370](#).

Figure 127: Edit Service Instance, Allowed Address Pair

Edit

Interface Type **Virtual Machine Interface**

left	(1.1.1.4) - f88f8960-7897-4df5-b...
right	(2.2.2.4) - 1fab8367-a4ed-43cb-...

▶ Service Health Check
 ▶ Routing Policy
 ▶ Route Aggregate

▼ **Allowed Address Pair**

Interface Type	IP	MAC	+
left	1.1.1.254	00:00:5e:00:01:03	+ -
right	2.2.2.254	00:00:5e:00:01:04	+ -

Cancel Save

s019908

For more information about OpenStack allowed address pairs, see

https://specs.openstack.org/openstack/neutron-specs/specs/api/allowed_address_pairs.html

Service Chain with Static Route Table

The service chain Version 2 also supports static route tables. A static route table is configured similar to how the allowed address pair is configured, except with using the label **right**. The route table will be attached to the correct VM ports of the port tuples, based on the configuration of the port with the label **right**.

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 “[Health Check Object](#)” on page 392.

Related Documentation

- [Health Check Object on page 392](#)

Using the Contrail Heat Template

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

- [Introduction to Heat on page 371](#)
- [Heat Architecture on page 371](#)
- [Support for Heat Version 2 Resources on page 371](#)
- [Heat Version 2 with Service Chaining and Port Tuple Sample Workflow on page 372](#)
- [Example: Creating a Service Template Using Heat on page 373](#)

Introduction to Heat

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

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

Heat templates can record the relationships between resources, for example, which networks are connected by means of policy enforcements, and consequently call OpenStack REST APIs that create the necessary infrastructure, in the correct order, needed to launch the application managed by the Heat template.

Heat Architecture

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

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

Support for Heat Version 2 Resources

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

OS::ContrailV2::<ResourceName>.

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

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

The **heat/** directory has the following subdirectories:

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

The following contains a list of all the generated plug-in resources that are supported by **contrail-heat** in Contrail Release 3.0.2 and greater:

<https://github.com/Juniper/contrail-heat/tree/master/generated/resources>

The following contains a list of new example templates:

https://github.com/Juniper/contrail-heat/tree/master/contrail_heat/new_templates

Deprecation of Heat Version 1 Resources

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

Heat Version 2 with Service Chaining and Port Tuple Sample Workflow

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

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

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

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

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

Example: Creating a Service Template Using Heat

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

1. Define a template to create the service template.

```
service_template.yaml
heat_template_version: 2013-05-23
description: >
  HOT template to create a service template
parameters:
  name:
    type: string
    description: Name of service template
  mode:
    type: string
    description: service mode
  type:
    type: string
    description: service type
  image:
    type: string
    description: Name of the image
  flavor:
    type: string
    description: Flavor
  service_interface_type_list:
    type: string
    description: List of interface types
  shared_ip_list:
    type: string
    description: List of shared ip enabled- disabled
  static_routes_list:
    type: string
    description: List of static routes enabled- disabled

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

```
}
```

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

```
service_template.env
```

```
parameters:
```

```
    name: contrail_svc_temp
```

```
    mode: transparent
```

```
    type: firewall
```

```
    image: cirros
```

```
    flavor: m1.tiny
```

```
    service_interface_type_list: management,left,right,other
```

```
    shared_ip_list: True,True,False,False
```

```
    static_routes_list: False,True,False,False
```

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

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

OR use this command for recent versions of OpenStack

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

Related Documentation

- [Service Chain Version 2 with Port Tuple on page 367](#)

Service Chain Route Reorigination

- [Overview: Service Chaining in Contrail on page 374](#)
- [Route Aggregation on page 376](#)
- [Routing Policy on page 382](#)
- [Control for Route Reorigination on page 390](#)

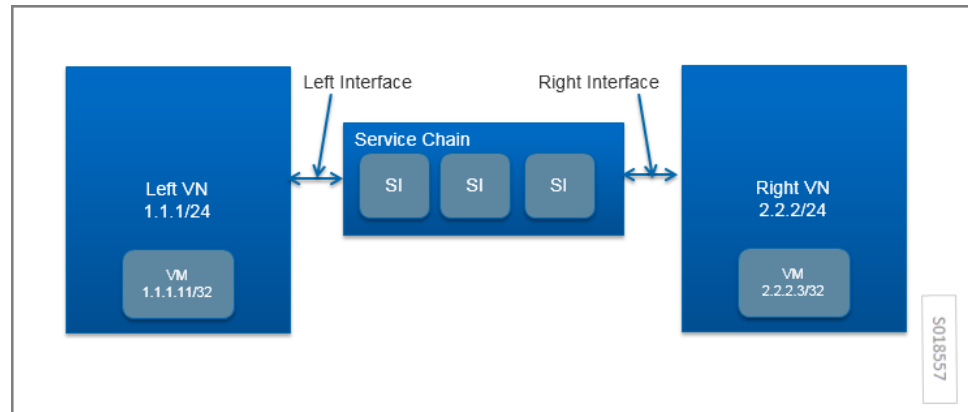
Overview: Service Chaining in Contrail

In Contrail, the service chaining feature allows the operator to insert dynamic services to control the traffic between two virtual networks. The service chaining works on a basic rule of next-hop stitching.

In [Figure 128 on page 375](#), the service chain is inserted between the Left VN and the Right VN. The service chain contains one or more service instances to achieve a required network policy.

In the example, the route for the VM in the Right VN is added to the routing table for the Left VN, with the next hop modified to ensure that the traffic is sent by means of the left interface of the service chain. This is an example of route reorigination.

Figure 128: Route Reorigination



Using reorigination of routes for service chaining (for example, putting the route for the right network in the left routing table) requires the following features:

- **Route aggregation**

For scaling purposes, it is useful to publish an aggregated route as the service chain route, rather than publishing every route of each VM (/32). This reduces the memory footprint for the route table in the gateway router and also reduces route exchanges between control nodes and the gateway router. The route can be aggregated to the default route (0/0), to the VN subnet prefix, or to any arbitrary route prefix.

- **Path attribute modification for reoriginated routes**

There are cases where the **BgpPath** attribute for the service chain route needs to be modified. An example is the case of service chain failover, in which there are two service chains with identical services that are connected between the same two VNs. The operator needs to control which service chain is used for traffic between two networks, in addition to ensuring redundancy and high availability by providing failover support. Path attribute modification for reoriginated routes is implemented by means of routing policy, by providing an option to alter the MED (multi-exit discriminator) or **local-pref** of the reoriginated service chain route.

- **Control to enable and disable reorigination of the route**

In some scenarios, the operator needs a control to stop reorigination of the route as the service chain route, for example, when static routes are configured on service VM interfaces. Control to enable or disable reorigination of the route is implemented by tagging the routes with the **no-reoriginate** community. Routes with the **no-reoriginate** community tag are skipped for route reorigination.

Route Aggregation

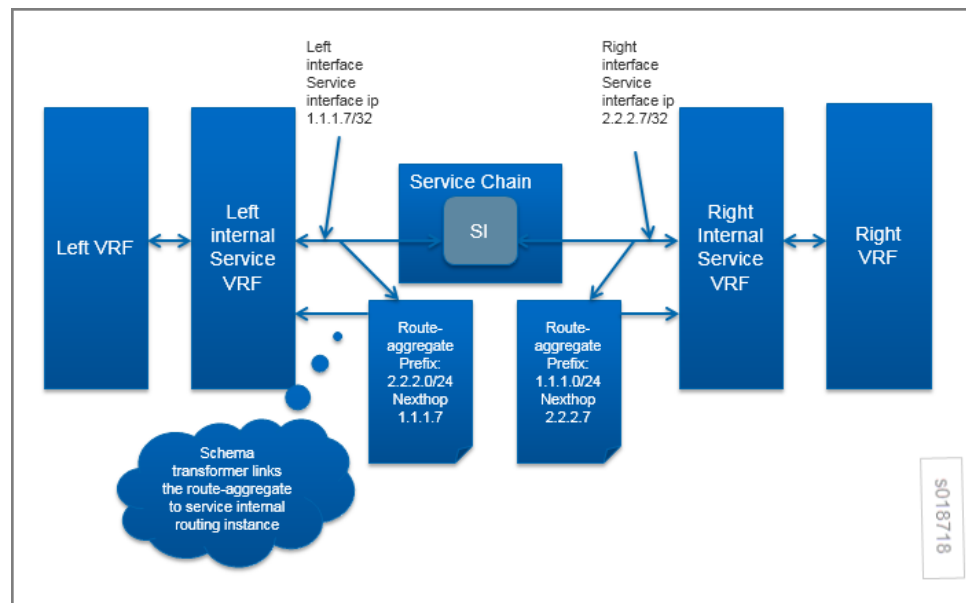
The route aggregation configuration object contains a list of prefixes to aggregate. The next-hop field in the route aggregate object contains the address of the route whose next hop is stitched as a next hop of the aggregate route.

Route aggregation is configured on the service instance. The operator can attach multiple route aggregation objects to a service instance. For example, if routes from the Right VN need to be aggregated and reoriginated in the route table of the Left VN, the route aggregate object is created with a prefix of the Right VN's subnet prefix and attached to the left interface of the service instance.

If the service chain has multiple service instances, the route aggregate object is attached to the left interface of the left-most service instance and to the right interface of the right-most service instance.

The relationships are shown in [Figure 129 on page 376](#).

Figure 129: Route Aggregate Relationships



The schema transformer sets the next-hop field of the route aggregate object to the service chain interface address. The schema transformer also links the route aggregate object to the internal routing instance created for the service instance.

Using the configuration as described, the Contrail control service reads the route aggregation object on the routing instance. When the first, more specific route or contributing route is launched (when the first VM is launched on the right VN), the aggregate route is published. Similarly, the aggregated route is deleted when the last, more specific route or contributing route is deleted (when the last VM is deleted in the right VN). The aggregated route is published when the next hop for the aggregated route gets resolved.

By default, in BGP or XMPP route exchanges, the control node will not publish contributing routes of an aggregate route.

Schema for Route Aggregation

- [Route Aggregate Object on page 377](#)
- [Service Instance Link to Route Aggregate Object on page 377](#)
- [Routing Instance Link to Route Aggregate Object on page 378](#)

Route Aggregate Object

The following is the schema for route aggregate objects. Multiple prefixes can be specified in a single route aggregate object.

```
<xsd:element name="route-aggregate" type="ifmap:IdentityType"/>
<xsd:complexType name="RouteListType">
  <xsd:element name="route" type="xsd:string" maxOccurs="unbounded"/>
</xsd:complexType>

<xsd:element name='aggregate-route-entries' type='RouteListType'/>
<!--#IFMAP-SEMANTICS-IDL
  Property('aggregate-route-entries', 'route-aggregate') -->

<xsd:element name='aggregate-route-nexthop' type='xsd:string'/>
<!--#IFMAP-SEMANTICS-IDL
  Property('aggregate-route-nexthop', 'route-aggregate') -->
```

Service Instance Link to Route Aggregate Object

The following is the schema for the service instance link to route aggregation objects. The operator can link multiple route aggregate objects to a single service interface.

```
<xsd:element name="route-aggregate" type="ifmap:IdentityType"/>
<xsd:complexType name="RouteListType">
  <xsd:element name="route" type="xsd:string" maxOccurs="unbounded"/>
</xsd:complexType>

<xsd:element name='aggregate-route-entries' type='RouteListType'/>
<!--#IFMAP-SEMANTICS-IDL
  Property('aggregate-route-entries', 'route-aggregate') -->

<xsd:element name='aggregate-route-nexthop' type='xsd:string'/>
<!--#IFMAP-SEMANTICS-IDL
  Property('aggregate-route-nexthop', 'route-aggregate') -->

<xsd:simpleType name="ServiceInterfaceType">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="management|left|right|other[0-9]*"/>
  </xsd:restriction>
</xsd:simpleType>
<xsd:complexType name='ServiceInterfaceTag'>
  <xsd:element name="interface-type" type="ServiceInterfaceType"/>
</xsd:complexType>

<xsd:element name="route-aggregate-service-instance"
type="ServiceInterfaceTag"/>
<!--#IFMAP-SEMANTICS-IDL
  Link('route-aggregate-service-instance',
```

```
'bgp:route-aggregate', 'service-instance', ['ref']) -->
```

Routing Instance Link to Route Aggregate Object

The following is the schema for the routing instance link to the route aggregation object. A routing instance can be linked to multiple route aggregate objects to perform route aggregation for multiple route prefixes.

```
<xsd:element name="route-aggregate-routing-instance"/>
<!--#IFMAP-SEMANTICS-IDL
    Link('route-aggregate-routing-instance',
        'route-aggregate', 'routing-instance', ['ref']) -->
```

Configuring and Troubleshooting Route Aggregation

- [Configure Route Aggregate Object on page 378](#)
- [Configuring a Service Instance on page 379](#)
- [Create a Virtual Network and Network Policy on page 379](#)
- [Validate the Route Aggregate Object in the API Server on page 380](#)
- [Validate the Route Aggregate Object in the Control Node on page 381](#)

Configure Route Aggregate Object

You can use the Contrail UI **Create Route Aggregate** screen to name the route aggregate object and identify the routes to aggregate. See [Figure 130 on page 378](#).

Figure 130: Create Route Aggregate

Example VNC Script to Create a Route Aggregate Object

You can use a VNC script to create a route aggregate object, as in the following example:

```
from vnc_api.vnc_api import *
vnc_lib = VncApi("admin", "<password>", "admin")
project=vnc_lib.project_read(fq_name=["default-domain", "admin"])
route_aggregate=RouteAggregate(name="left_to_right", parent_obj=project)
route_list=RouteListType(["<ip address>"])
```

```
route_aggregate.set_aggregate_route_entries(route_list)
vnc_lib.route_aggregate_create(route_aggregate)
```

Configuring a Service Instance

Create a service instance with the route aggregate object linked to the aggregate left network subnet prefix in the right virtual network. See the example in [Figure 131 on page 379](#).

Figure 131: Create Service Instance

Create Service Instance

si-aggregate st-with-aggregate - [transparent (left, right)...]

▼ Interface Details

Interface Type Virtual Network

left Auto Configured

right Auto Configured

▼ Advanced Options

Routing Policy

Route Aggregate

Interface Type Route Aggregate

right left-to-right

Cancel Save

Create a Virtual Network and Network Policy

Create a left and right virtual network with the subnets 1.1.1/24 and 2.2.2/24, respectively. Create a network policy to apply a service chain between the left VN and the right VN. See the following example.

Create Policy

Policy Name

service-chain-policy

Policy Rules

Action	Protocol	Source	Ports	Direction	Destination	Ports	Log	Services Mirror
PASS	ANY	left	ANY	left to right	right	ANY	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

Service Instance

si-aggregate

+ Add Rule

Cancel Save

Attach the network policy to create the service chain between the left and right VN. See the following example.

Validate the Route Aggregate Object in the API Server

Validate the route aggregate object in the API server configuration database. Verify the routing instance reference and the service instance reference for the aggregate object. The **aggregate_route_nexthop** field in the route aggregate object is initialized by the schema transformer to the service chain address. See the following example.

```
{
  - route-aggregate: {
    - fq_name: {
      "default-domain",
      "admin",
      "left-to-right"
    },
    uuid: "872b1fbd-b36c-4165-8723-7e10806d7716",
    parent_uuid: "6861d89d-a02f-4215-b329-1864084c8a75",
    aggregate_route_nexthop: "1.1.1.3",
    - routing_instance_refs: {
      - {
        - to: {
          "default-domain",
          "admin",
          "right",
          "service-ace7ae00-56e3-42d1-96ec-7fe7708d97f-default-domain_admin_si-aggregate"
        },
        href: "http://nodes27.englab.juniper.net:8082/routing-instance/d291a95a-1a5a-4fce-94c8-4abd0968d992",
        attr: null,
        uuid: "d291a95a-1a5a-4fce-94c8-4abd0968d992"
      }
    },
    parent_href: "http://nodes27.englab.juniper.net:8082/project/6861d89d-a02f-4215-b329-1864084c8a75",
    parent_type: "project",
    perms2: {(-)},
    href: "http://nodes27.englab.juniper.net:8082/route-aggregate/872b1fbd-b36c-4165-8723-7e10806d7716",
    + id_perms: {(-)},
    - aggregate_route_entries: {
      - route: {
        "1.1.1.0/24"
      }
    },
    display_name: "left-to-right",
    - service_instance_refs: {
      - {
        - to: {
          "default-domain",
          "admin",
          "si-aggregate"
        },
        href: "http://nodes27.englab.juniper.net:8082/service-instance/62accf30-8cc8-4148-b7b8-975573b0d950",
        attr: {
          interface_type: "right"
        },
        uuid: "62accf30-8cc8-4148-b7b8-975573b0d950"
      }
    },
    name: "left-to-right"
  }
}
```

Validate the Route Aggregate Object in the Control Node

Validate the instance configurations of the route aggregate by checking the control node introspect for the service instance internal routing instance. For example:

http://<control-node>:8083/Snh_ShowBgpInstanceConfigReq?search_string=default-domain:admin:right:service-ace7ae00-56e3-42d1-96ec-7fe77088d97f-default-domain_admin_si-aggregate

See the following example.

service_chain_infos					static_routes		aggregate_routes	
family	routing_instance	chain_address	prefixes	service_instance	static_routes	aggregate_routes	prefix	nexthop
inet	default-domain:admin:left:left	1.1.1.3	1.1.1.0/24	default-domain:admin:si-aggregate			1.1.1.0/24	1.1.1.3

To check the state of the route aggregate object on the control node, point your browser to:

http://<control-node>:8083/Snh_ShowRouteAggregateReq

See the following example.

service_chain_infos					static_routes		aggregate_routes	
family	routing_instance	chain_address	prefixes	service_instance	static_routes	aggregate_routes	prefix	nexthop
inet	default-domain:admin:left:left	1.1.1.3	1.1.1.0/24	default-domain:admin:si-aggregate			1.1.1.0/24	1.1.1.3

You can also check the route table for the aggregate route in the right VN BGP able. For example:

http://<control-node>:8083/Snh_ShowRouteReq?x=default-domain:admin:right:right.inet.0

See the following example.

routes									
prefix	last_modified	paths							
1.1.1.0/24	2016-Feb-18 05:00:29.215876								
protocol	last_modified	local_preference	local_as	peer_as	peer_router_id	source_as	path	next_hop	label
Aggregate	2016-Feb-18 05:00:29.215876	100	0	0	-	-	-	10.204.216.23	22

Routing Policy

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

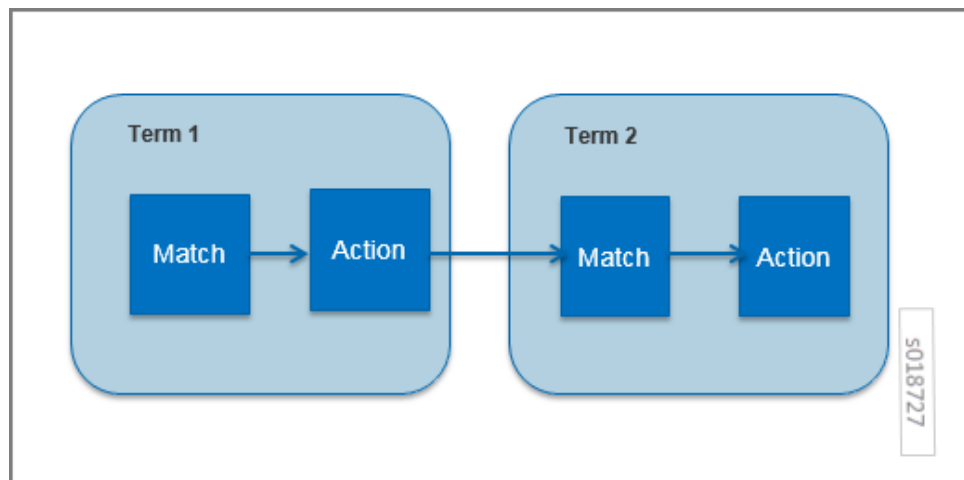
The routing policy contains list terms. A term can be a terminal rule, meaning that upon a match on the specified term, no further terms are evaluated and the route is dropped or accepted, based on the action in that term.

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.

Match Condition	User Input	Description
Prefix	List of prefixes to match	<p>Each prefix in the list is represented as prefix and match type, where the prefix match type can be:</p> <ul style="list-style-type: none"> • exact • orlonger • longer <p>Example: 1.1.0.0/16 orlonger</p> <p>A route matches this condition if its prefix matches any of the prefixes in the list.</p>
Community	Community string to match	<p>Represented as either a well-known community string with no export or no reoriginate, or a string representation of a community (64512:11).</p>
Protocol	Array of path source or path protocol to match	<p>BGP XMPP StaticRoute ServiceChain Aggregate. A path is considered as matching this condition if the path protocol is one of protocols in the list.</p>

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.

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.
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.

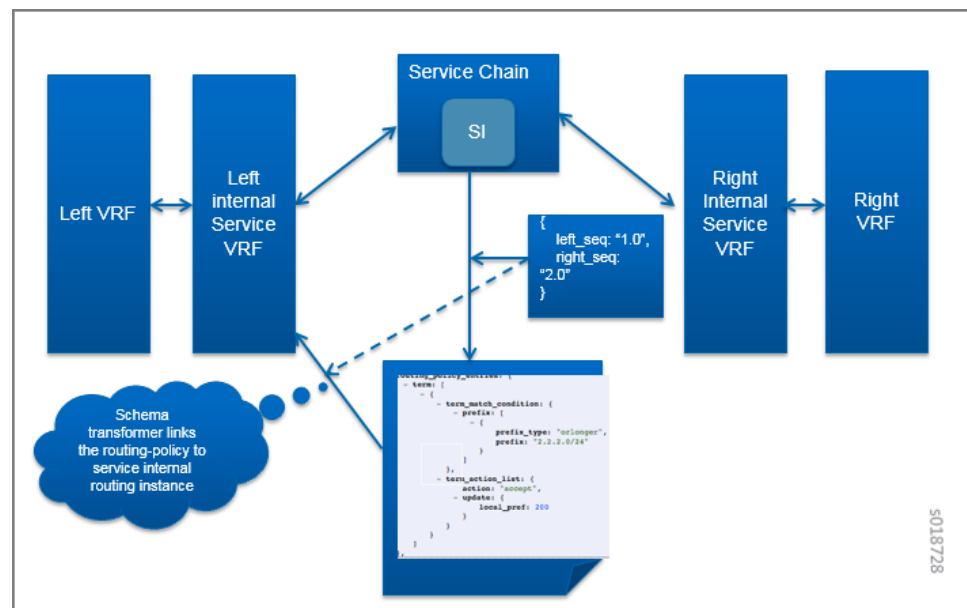
Update Action	User Input	Description
community	List of community	As part of the policy update, the following actions can be taken for community: <ul style="list-style-type: none"> • Add a list of community to the existing community. • Set a list of community. • Remove a list of community (if present) 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

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.



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 on page 386](#)
- [Configure Service Instance on page 387](#)
- [Configure the Network Policy for the Service Chain on page 387](#)

Create Routing Policy

First, create the 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

Cancel Save

s018729

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.

Configure the Network Policy for the Service Chain

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

Action	Protocol	Source	Ports	Direction	Destination	Ports	Log	Services	Mirror
PASS	ANY	left			right				

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"
-----
s018732

```

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	terminal	matches	actions	false
			true	matches	actions	
				prefix [2.2.2.0/24 orlonger]	accept local-pref 200	
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 }

5018733

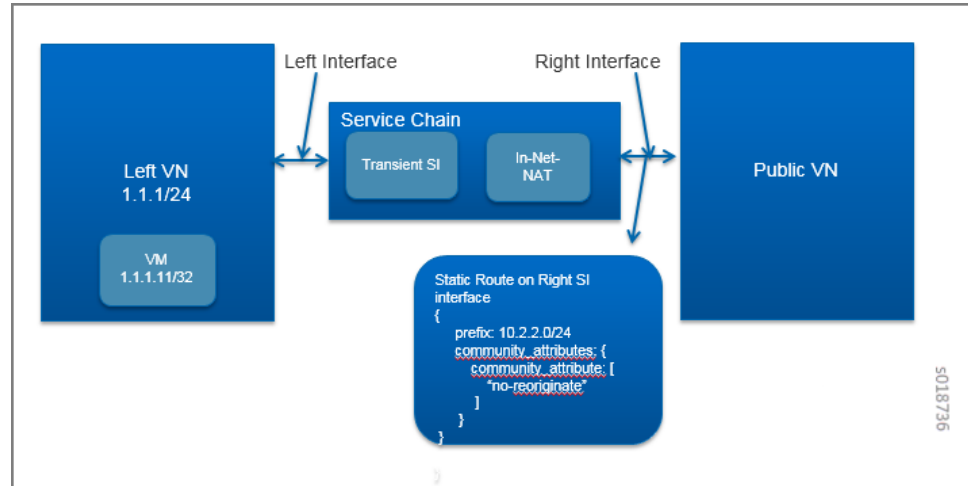
Verify Routing Policy Configuration on the Routing Instance

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>

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



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.

Create Service Instance

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**

+ Add Static Routes

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

Routing Policy

Interface Type: **left**

Routing Policy: **no-reoriginate**

Cancel Save

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_bref: "http://nodeq2:8082/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

Health Check Object

- [Health Check Overview on page 392](#)
- [Health Check Object Configuration on page 392](#)
- [Creating a Health Check with the Contrail User Interface on page 394](#)
- [Using the Health Check on page 395](#)
- [Health Check Process on page 395](#)

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 42 on page 393](#) shows the configurable properties of the health check object.

Table 42: 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: In versions prior to Contrail 4.1, end-to-end health check is not supported on a transparent service chain. However, a link-local health check is possible on a transparent service instance if the corresponding service instance interface is configured with its IP address. Contrail 4.1 supports a segment-based health check for transparent service chain.

Creating a Health Check with the Contrail User Interface

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 132 on page 394](#).

Figure 132: Create Health Check Screen

The screenshot shows a 'Create' dialog box for a 'Health Check Service'. It has two tabs: 'Health Check Service' (selected) and 'Permissions'. The form contains the following fields:

- Name:** A text input field containing 'ext_hc_service'.
- Protocol:** A dropdown menu with 'PING' selected.
- Monitor Target:** A text input field containing '8.8.8.8'.
- Delay (secs):** A text input field containing '3'.
- Timeout (secs):** A text input field containing '5'.
- Retries:** A text input field containing '2'.
- Health Check Type:** A dropdown menu with 'End-To-End' selected.

At the bottom right, there are 'Cancel' and 'Save' buttons. A vertical text '5018766' is visible on the right edge of the dialog box.

- 2. Complete the fields to define the permissions for the health check, see [Table 43 on page 394](#).

Table 43: 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.

Table 43: Create Health Check Fields (*continued*)

Field	Description
Timeout (secs)	The number of seconds to wait for a response.
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.

Examples: Configuring Service Chaining

- [Example: Creating an In-Network or In-Network-NAT Service Chain on page 397](#)
- [Example: Creating a Transparent Service Chain on page 405](#)
- [Example: Creating a Service Chain With the CLI on page 409](#)

Example: Creating an In-Network or In-Network-NAT Service Chain

This section provides an example of creating an **in-network** service chain and an **in-network-nat** service chain using the Juniper Networks Contrail user interface. This service chain example also shows scaling of service instances.

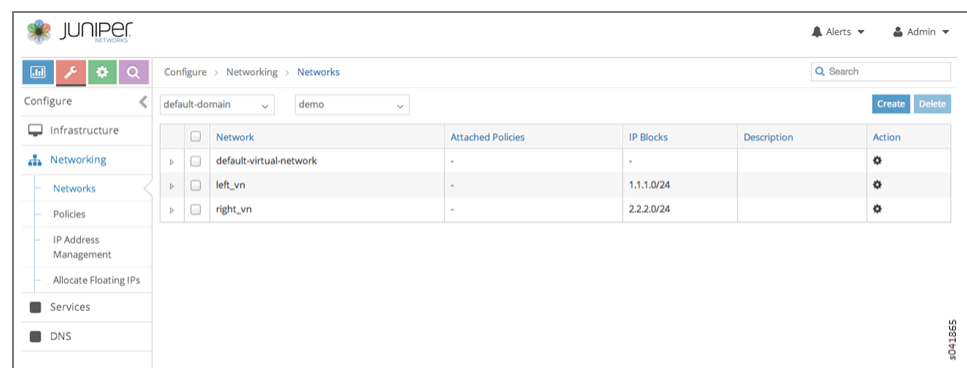
- [Creating an In-Network or In-Network-NAT Service Chain on page 397](#)

Creating an In-Network or In-Network-NAT Service Chain

To create an **in-network** or **in-network-nat** service chain:

1. Create a left and a right virtual network. Select **Configure > Networking > Networks** and create **left_vn** and **right_vn**; see [Figure 133 on page 397](#).

Figure 133: Create Networks



2. Configure a service template for an in-network service template for NAT. Navigate to **Configure > Services > Service Templates** and click the **Create** button on **Service Templates**. The **Add Service Template** window appears; see [Figure 134 on page 398](#).

Figure 134: Add Service Template

Add Service Template

Name: nat-template

Service Mode: In-Network

Image Name: nat-service

Interface Types	Shared IP	Static Routes	+
Management	<input type="checkbox"/>	<input type="checkbox"/>	+ -
Left	<input checked="" type="checkbox"/>	<input type="checkbox"/>	+ -
Right	<input type="checkbox"/>	<input type="checkbox"/>	+ -

▼ Advanced options

Service Scaling: ☒

Instance Flavor: m1.medium(RAM:4096, CPU cores:2, Disk:...)

Cancel Save

s041902

Table 44: Add Service Template Fields

Field	Description
Name	Enter a name for the service template.
Service Mode	Select the service mode: In-Network (for firewall service), In-Network-NAT (for NAT service), or Transparent .
Service Scaling	If you will be using multiple virtual machines for a single service instance to scale out the service, select the Service Scaling check box. When scaling is selected, you can choose to use the same IP address for a particular interface on each virtual machine interface or to allocate new addresses for each virtual machine. For a NAT service, the left (inner) interface should have the same IP address, and the right (outer) interface should have a different IP address.
Image Name	Select from a list of available images the image for the service. NOTE: Only images that have been tagged as public in Glance will appear in the drop-down list.

Table 44: Add Service Template Fields (*continued*)

Field	Description
Interface Types	<p>Select the interface type or types for this service:</p> <ul style="list-style-type: none"> For firewall or NAT services, both Left Interface and Right Interface are required. For an analyzer service, only a Left Interface is required. For Juniper Networks virtual images, Management Interface is also required, in addition to any left or right requirement.

3. On **Add Service Template**, complete the following for the in-network service template:

- **Name:** nat-template
- **Service Mode:** In-Network
- **Service Scaling:** Select from Advanced
- **Image Name:** nat-service
- **Interface Types:** Select Left Interface and Right Interface. For Juniper Networks virtual images, select Management Interface as the first interface.
- The Left Interface will be automatically marked for sharing the same IP address

4. If multiple instances are to be launched for a particular service instance, select the **Service Scaling** check box, which enables the **Shared IP** feature. [Figure 135 on page 400](#) shows the **Left** interface selected, with the **Shared IP** check box selected, so the left interface will share the IP address.



NOTE: The **Shared IP** for **Service Scaling** is an internal infrastructure feature used only for service scaling, it cannot be used for other features.

Figure 135: Add Service Template Shared IP

Add Service Template

Name

Service Mode **Service Type**

Service Scaling ☒

Image Name

Interface Types **Shared IP** ☒ + -

Service Interface	Shared IP
Management	Disabled
Right	Disabled
Left	Enabled

Cancel **Save**

- Click **Save**.

The service template is created and appears on the **Service Templates** screen, see [Figure 136 on page 400](#).

Figure 136: Service Templates

Configure > Services > Service Templates

default-domain

Create **Delete**

Template	Service Mode	Service Scaling	Interfaces	Image Name	Flavor
<input type="checkbox"/> nat-template	In-network	Enabled	Management, Left(Shared IP), Right	nat-service	m1.medium

- Create the service instance. Navigate to **Configure > Services > Service Instances**, and click **Create**, then select the template to use and select the corresponding left, right, or management networks; see [Figure 137 on page 401](#).

Figure 137: Create Service Instances

Table 45: Create Service Instances Fields

Field	Description
Instance Name	Enter a name for the service instance.
Services Template	Select from a list of available service templates the service template to use for this instance.
Number of Instances	If scaling is enabled, enter a value in the Number of Instances field to define the number of instances of service virtual machines to launch.
Interface List and Virtual Networks	An ordered list of interfaces as defined in the Service Template. If you are using the Management Interface , select Auto Configured . The software will use an internally-created virtual network. For Left Interface , select left_vn and for Right Interface , select right_vn .

- If static routes are enabled for specific interfaces, open the **Static Routes** field below each enabled interface and enter the static route address details; see [Figure 138 on page 402](#).

Figure 138: Create Service Instances

Create Service Instances

Instance Name:

Services Template: nat-ecmp-template - [in-network (management, left, right)]

Number of instances: 1

Interface 1: Management Auto Configured

Interface 2: Left vn10 (admin)

▼ Static Routes

Prefix	Next hop	
10.204.80.0/28	Interface 2	+ -

Interface 3: Right vn10 (admin)

Cancel Save

- The console for the service instances can be viewed. At **Configure > Services > Service Instances**, click the arrow next to the name of the service instance to reveal the details panel for that instance, then click **View Console** to see the console details; see [Figure 139 on page 402](#) and [Figure 140 on page 403](#).

Figure 139: Service Instance Details

fw-instance firewall-template (Transparent) Active 1 Instances Management Network: Automatic Left Network: Automatic Right Network: Automatic

Instance Name: fw-instance

Template: firewall-template (Transparent)

Number of instances: 1 Instances

Networks: Management Network: Automatic, Left Network: Automatic, Right Network: Automatic

Image: m1.medium

Flavor: vcnbridge

Instance Details

Virtual Machine	Status	Power State	Networks
fw-instance_1	ACTIVE	RUNNING	svc-vn-mgmt250.250.1.252 svc-vn-left250.250.2.253 svc-vn-right250.250.3.253

Static Route

[View Console](#)

Figure 140: Service Instance Console

0.204.216.36:5999/vnc_auto.html?token=9eada783-24e7-4808-9325-4ad257bf3762

Connected (unencrypted) to: QEMU (instance-0000000b)

Interface	Admin	Link	Proto	Local	Remote
ge-0/0/0	up	up			
ge-0/0/0.0	up	up	inet	250.250.1.253/24	
ge-0/0/0	up	up			
ip-0/0/0	up	up			
lsq-0/0/0	up	up			
lt-0/0/0	up	up			
mt-0/0/0	up	up			
sp-0/0/0	up	up			
sp-0/0/0.0	up	up	inet	10.0.0.1	--> 10.0.0.16
sp-0/0/0.16383	up	up	inet	10.0.0.6	--> 0/0
				128.0.0.1	--> 128.0.1.16
				128.0.0.6	--> 0/0
ge-0/0/1	up	up			
ge-0/0/1.0	up	up	inet	1.1.1.253/24	
ge-0/0/2	up	up			
ge-0/0/2.0	up	up	inet	2.2.2.253/24	
dsc	up	up			
gre	up	up			
ipip	up	up			
lo0	up	up			
lo0.16384	up	up	inet	127.0.0.1	--> 0/0
lo0.16385	up	up	inet	10.0.0.1	--> 0/0

(more)					

5041919

9. Configure the network policy. Navigate to **Configure > Networking > Policies**.

- Name the policy and associate it with the networks created earlier: **left_vn** and **right_vn**.
- Set source network as **left_vn** and destination network as **right_vn**.
- Select **Apply Service** and select the service (**nat-ecmp**).

Figure 141: Create Policy

Create Policy

Policy Name
fw-policy

Policy Rules

Action	Protocol	Source Network	Source Ports	Direction	Destination Network	Destination Ports	Apply Service	Mirror to
PAS	ANY	left_vn	Source	<>	right_vn	Destin	<input checked="" type="checkbox"/>	<input type="checkbox"/>

fw-instance x

Cancel Save

5041870

10. Associate the policy with both the **left_vn** and the **right_vn**. Navigate to **Configure > Networking > Network**.

- On the right side of **left_vn**, click the gear icon to enable **Edit Network**.
- In the **Edit Network** dialog box for **left_vn**, select **nat-policy** in the **Network Policy(s)** field.
- Repeat the same process for the **right_vn**.

Figure 142: Edit Network

Network Name:

Network Policy(s):

Address Management: IP Block: Gateway:

IPAM	IP Block	Gateway
default-domain:default-project:default-network-ipam	1.1.1.0/24	1.1.1.254

Route Targets

Floating IP Pools

Host Routes

Advanced Options

Cancel Save

11. Launch virtual machines (from OpenStack) and test the traffic through the service chain by doing the following:
 - a. Navigate to **Configure > Networking > Policies**.
 - b. Launch **left_vm** in virtual network **left_vn**.
 - c. Launch **right_vm** in virtual network **right_vn**.
 - d. Ping from **left_vm** to **right_vm** IP address (**2.2.2.252** in Figure 143 on page 404).
 - e. A **TCPDUMP** on the **right_vm** should show that packets are NAT-enabled and have the source IP set to **2.2.2.253**.

Figure 143: Launch Instances

Instances

Launch Instance Terminate Instances

Instance Name	IP Address	Size	Keypair	Status	Task	Power State	Actions
nat-instance_1	250.250.1.253	m1.medium 4GB RAM 2 VCPU 40GB Disk	-	Active	None	Running	Create Snapshot More
right_vm	2.2.2.252	m1.tiny 512MB RAM 1 VCPU 0 Disk	-	Active	None	Running	Create Snapshot More
left_vm	1.1.1.252	m1.tiny 512MB RAM 1 VCPU 0 Disk	-	Active	None	Running	Create Snapshot More

Displaying 3 items

- Related Documentation**
- [Service Chaining on page 357](#)
 - [Example: Creating a Transparent Service Chain on page 405](#)
 - [ECMP Load Balancing in the Service Chain on page 362](#)

Example: Creating a Transparent Service Chain

This section provides an example of creating a transparent mode service chain using the Juniper Networks Contrail user interface. Also called bridge mode, transparent mode is used for services that do not modify the packet, such as Layer 2 firewall, Intrusion Detection and Prevention (IDP), and so on. The following service chain example also shows scaling of service instances.

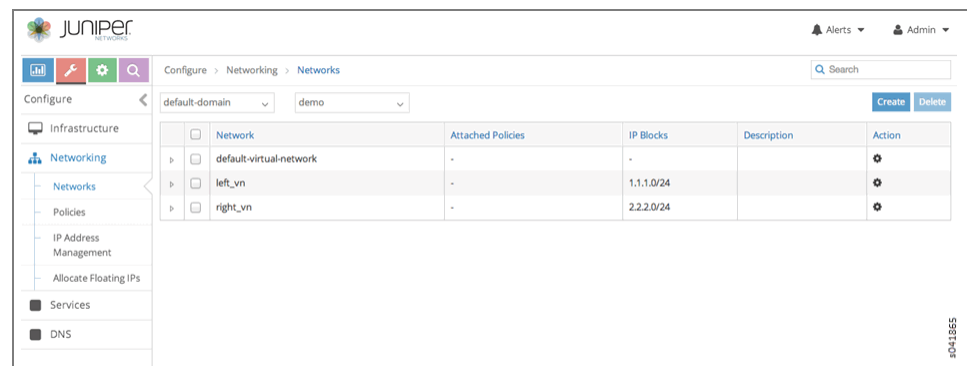
- [Creating a Transparent Mode Service Chain on page 405](#)

Creating a Transparent Mode Service Chain

To create a transparent mode service chain:

1. Create a left and a right virtual network. Select **Configure > Networking > Networks** and create **left_vn** and **right_vn**; see [Figure 144 on page 405](#).

Figure 144: Create Networks



2. Configure a service template for a transparent mode. Navigate to **Configure > Services > Service Templates** and click the **Create** button on **Service Templates**. The **Add Service Template** window appears; see [Figure 145 on page 406](#).

Figure 145: Add Service Template

Add Service Template

Name:

Service Mode:

Image Name:

Interface Types	Shared IP	Static Routes	+
Management	<input type="checkbox"/>	<input type="checkbox"/>	+ -
Left	<input checked="" type="checkbox"/>	<input type="checkbox"/>	+ -
Right	<input checked="" type="checkbox"/>	<input type="checkbox"/>	+ -

Advanced options

Service Scaling: ☒

Instance Flavor:

Cancel Save

Table 46: Add Service Template Fields

Field	Description
Name	Enter a name for the service template.
Service Mode	Select the service mode: In-Network or Transparent .
Service Scaling	If you will be using multiple virtual machines for a single service instance to scale out the service, select the Service Scaling check box. When scaling is selected, you can choose to use the same IP address for a particular interface on each virtual machine interface or to allocate new addresses for each virtual machine. For a NAT service, the left (inner) interface should have the same IP address, and the right (outer) interface should have a different IP address.
Image Name	Select from a list of available images the image for the service.
Interface Types	Select the interface type or types for this service: <ul style="list-style-type: none"> For firewall or NAT services, both Left Interface and Right Interface are required. For an analyzer service, only Left Interface is required. For Juniper Networks virtual images, Management Interface is also required, in addition to any left or right requirement.

3. On **Add Service Template**, complete the following for the transparent mode service template:

- **Name:** firewall-template
- **Service Mode:** Transparent
- **Service Scaling:** Select this.
- **Image Name:** vsrx-bridge
- **Interface Types:** Select Left Interface, Right Interface, and Management Interface.

If multiple instances are to be launched for a particular service instance, select the **Service Scaling** check box, which enables the **Shared IP** feature.

4. Click **Save**.

5. Create the service instance. Navigate to **Configure > Services > Service Instances**, and click **Create**, then select the template to use and select the corresponding left, right, or management networks; see [Figure 146 on page 407](#).

Figure 146: Create Service Instances

Table 47: Create Service Instances Fields

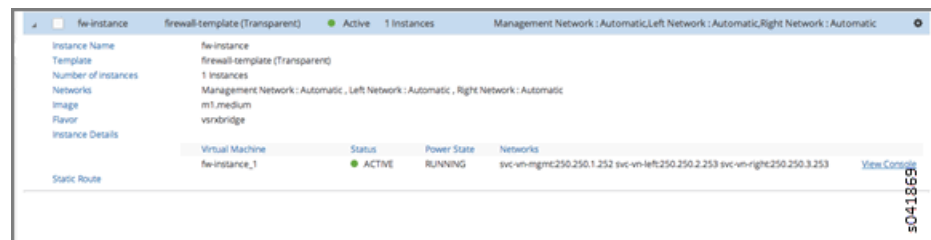
Field	Description
Instance Name	Enter a name for the service instance.
Services Template	Select from a list of available service templates the service template to use for this instance.
Left Network	Select from a list of available virtual networks the network to use for the left interface. For transparent mode, select Auto Configured .

Table 47: Create Service Instances Fields (*continued*)

Field	Description
Right Network	Select from a list of available virtual networks the network to use for the right interface. For transparent mode, select Auto Configured
Management Network	If you are using the Management Interface , select Auto Configured . The software will use an internally-created virtual network. For transparent mode, select Auto Configured

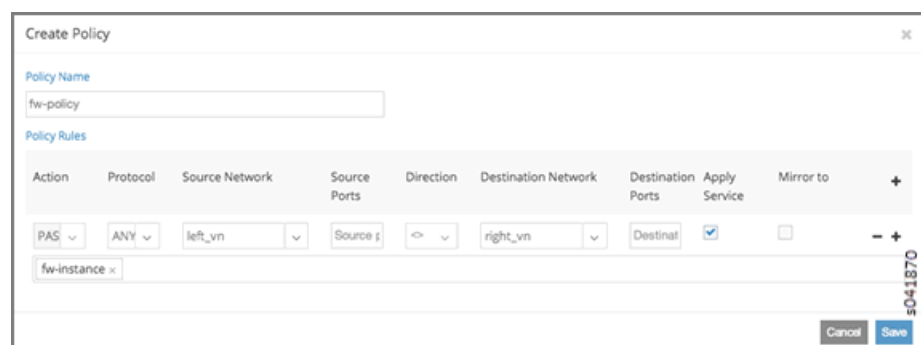
6. If scaling is enabled, enter a value in the **Number of Instances** field to define the number of instances of service virtual machines to launch; see [Figure 147 on page 408](#).

Figure 147: Service Instance Details



7. Next, configure the network policy. Navigate to **Configure > Networking > Policies**.
- Name the policy **fw-policy**.
 - Set source network as **left_vn** and destination network as **right_vn**.
 - Check **Apply Service** and select the service (**fw-instance**).

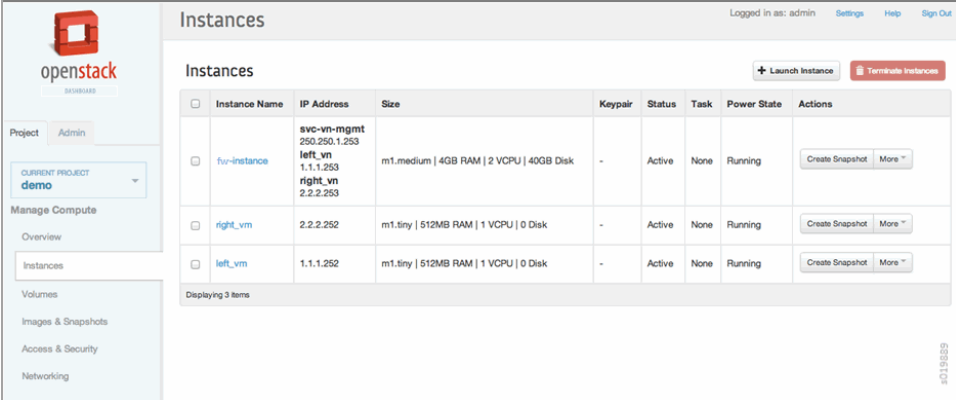
Figure 148: Create Policy



8. Next, associate it to the networks created earlier – **left_vn** and **right_vn**. Navigate to **Configure > Networking > Policies**.
- On the right side of **left_vn**, click the gear icon to enable **Edit Network**.
 - In the **Edit Network** dialog box for **left_vn**, select **nat-policy** in the **Network Policy(s)** field.

- Repeat the process for the **right_vn**.
- 9. Next, launch virtual machines (from OpenStack) and test the traffic through the service chain by doing the following:
 - a. Navigate to **Configure > Networking > Policies**.
 - b. Launch **left_vm** in virtual network **left_vn**.
 - c. Launch **right_vm** in virtual network **right_vn**.
 - d. Ping from **left_vm** to **right_vm** IP address (2.2.2.252 in [Figure 149 on page 409](#)).
 - e. A **TCPDUMP** on the **right_vm** should show that packets have the source IP set to 2.2.2.253.

Figure 149: Launch Instances



Instance Name	IP Address	Size	Keypair	Status	Task	Power State	Actions
svc-vn-mgmt 250.250.1.253	250.250.1.253	m1.medium 4GB RAM 2 VCPU 40GB Disk	-	Active	None	Running	Create Snapshot More
left_vm 1.1.1.253	1.1.1.253	m1.tiny 512MB RAM 1 VCPU 0 Disk	-	Active	None	Running	Create Snapshot More
right_vm 2.2.2.252	2.2.2.252	m1.tiny 512MB RAM 1 VCPU 0 Disk	-	Active	None	Running	Create Snapshot More

Related Documentation • [Service Chaining on page 357](#)

Example: Creating a Service Chain With the CLI

This section provides syntax and examples for creating service chaining objects for Contrail Controller.

- [CLI for Creating a Service Chain on page 409](#)
- [CLI for Creating a Service Template on page 410](#)
- [CLI for Creating a Service Instance on page 410](#)
- [CLI for Creating a Service Policy on page 410](#)
- [Example: Creating a Service Chain with VSRX and In-Network or Routed Mode on page 411](#)

CLI for Creating a Service Chain

All of the commands needed to create service chaining objects are located in **/opt/contrail/utls**.

CLI for Creating a Service Template

The following commands are used to create a service template:

```
./service-template.py add    [--svc_type {firewall, analyzer}]  
  
                             [--image_name IMAGE_NAME]  
  
                             template_name  
  
./service-template.py del    template_name
```

CLI for Creating a Service Instance

The following commands are used to create a service instance:

```
./service-instance.py add    [--proj_name PROJ_NAME]  
  
                             [--mgmt_vn MGMT_VN]  
  
                             [--left_vn LEFT_VN]  
  
                             [--right_vn RIGHT_VN]  
  
                             instance_name  
  
                             template_name  
  
./service-instance.py del    [--proj_name PROJ_NAME]  
  
                             instance_name  
  
                             template_name
```

CLI for Creating a Service Policy

The following commands are used to create a service policy:

```
./service-policy.py add      --svc_list SVC_LIST [SVC_LIST ...]  
  
                             --vn_list VN_LIST [VN_LIST ...]  
  
                             [--proj_name PROJ_NAME]  
  
                             policy_name  
  
./service-policy.py del      [--proj_name PROJ_NAME]  
  
                             policy_name
```

Example: Creating a Service Chain with VSRX and In-Network or Routed Mode

The following example creates a VSRX firewall service in a virtual network named **test**, using a project named **demo** and a template, an instance, and a policy, all named **test**.

1. Add images to Glance (OpenStack image service).

- a. Download the following images:

```
precise-server-cloudimg-amd64-disk1.img
```

```
junos-vsrx-12.1-nat.img
```

- b. Add the images to Glance, using the names **ubuntu** and **vsrx**.

```
(source /etc/contrail/openstackrc; glance add name='ubuntu' is_public=true
container_format=ovf disk_format=qcow2 <
precise-server-cloudimg-amd64-disk1.img)
```

```
(source /etc/contrail/openstackrc; glance add name='vsrx' is_public=true
container_format=ovf disk_format=qcow2 < junos-vsrx-12.1-dhcp.img)
```

2. Create a service template of type **firewall** and named **vsrx**.

```
./service-template.py add test_template --svc_type firewall --image_name vsrx
```

3. Create virtual networks.

```
VN1
```

```
VN2
```

4. Create a service template.

```
./service-template.py add --svc_scaling ecmp-template
```

5. Create a service instance.

```
./service-instance.py add --proj_name admin --left_vn VN1 --right_vn VN2
--max_instances 3 ecmp-instance ecmp-template
```

6. Create a service policy.

```
./service-policy.py add proj_name admin --svc_list ecmp-instance --vn_list VN1 VN2
ecmp-policy
```

7. Create virtual machines and attach them to virtual networks.

```
VM1 (attached to VN1)—use ubuntu image
```

```
VM2 (attached to VN2)—use ubuntu image
```

8. Launch the instances **VM1** and **VM2**.

9. Send ping traffic from **VM1** to **VM2**.

10. Send traffic from **VM1** in **VN1** to **VM2** in **VN2**.
11. You can use the Contrail Juniper Networks interface to monitor the ping traffic flows. Select **Monitor > Infrastructure > Virtual Routers** and select an individual vRouter. Click through to view the vRouter details, where you can click the **Flows** tab to view the flows.

Related Documentation

- [Service Chaining on page 357](#)

Adding Physical Network Functions in Service Chains

- [Using Physical Network Functions in Contrail Service Chains on page 413](#)
- [Example: Adding a Physical Network Function Device to a Service Chain on page 414](#)

Using Physical Network Functions in Contrail Service Chains

Contrail Release 3.0 and greater supports service appliance-based physical network functions devices (PNFs) in service chains, enabling the creation of service chains that include a combination of virtual network functions (VNFs) and PNFs. The PNFs are also supported with Contrail Device Manager.

- [PNF Service Chaining Objects on page 413](#)
- [Prerequisites and Assumptions on page 414](#)

PNF Service Chaining Objects

As of Contrail Release 3.0, Contrail has objects used to support PNFs in service chains, including:

- service appliance (SA)—represents a single physical appliance
- service appliance set (SA set)—represents a collection of functionally equivalent SAs, all running the same software with the same capabilities

A service appliance is associated with a physical router that has physical interfaces for the left, right, management, or other interfaces.

There can be more than one service appliance and associated physical router and physical interface objects representing it.

A physical appliance can host more than one service appliance through a logical system or other virtualization capability.

The service template object supports a physical network function service template (PNF-ST). The PNF-ST is associated with a service appliance set, which represents a pool of service appliances that can be used when the PNF-ST is instantiated.

Only the transparent service mode is supported for PNF-STs.

Prerequisites and Assumptions

The following are the prerequisites for implementing a service appliance with a Contrail controller.

- Before the controller can use a PNF SA, the controller must be connected to a service control gateway (SCG) router, such as an MX Series router.
- The Contrail Device Manager must manage the SCG router.
- The PNF SA must be configured, and it must be configured to operate as an Ethernet bridge. The Contrail controller does not automatically implement PNF SA configuration.
- Infrastructure interfaces (physical interfaces or aggregated Ethernet interfaces) on the SCG facing the SA must be preconfigured. The interfaces must be able to support VLAN-based units.
- Layer 2 VPNs as supported by the Contrail Device Manager are only available with Juniper Networks Junos Release 14.2R4 or greater. If an earlier version of Junos is used, you must mark the virtual networks in Contrail with the forwarding mode “L3 only.”
- Logical interfaces for connecting the service gateway VRFs to the customer and the Internet must be preconfigured.

Related Documentation

- [Example: Adding a Physical Network Function Device to a Service Chain on page 414](#)

Example: Adding a Physical Network Function Device to a Service Chain

Beginning with Contrail 3.0, it is possible to add a physical network function (PNF) device to a service chain. This section provides an example of creating a service chain that includes a PNF.

- [Prerequisites for Adding a PNF to a Service Chain on page 415](#)

Prerequisites for Adding a PNF to a Service Chain

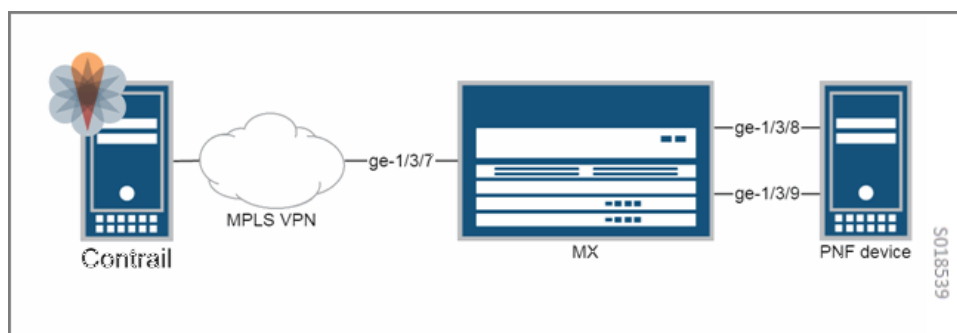
Prerequisites The following are the minimum requirements needed before you can add a PNF to a service chain using the procedure shown in the example included in this topic:

- at least one MX Series device
- at least one PNF to connect to the MX device
- Juniper Networks Junos version that includes the feature **accept-local-nexthop**



NOTE: The Junos feature **accept-local-nexthop** is available starting with Junos Release 14.1X55. The Contrail service chain with PNF has been tested on Junos 14.1X55. Contact your Juniper Networks customer service representative for more information.

The prerequisite minimum topology is shown in the following figure.



The following must be preconfigured on the MX Series device.

```

interfaces {
  ge-1/3/7 {
    unit 0 {
      family inet {
        address 10.227.5.115/24;
      }
      family mpls;
    }
  }
  ge-1/3/9 {
    vlan-tagging;
  }
  ge-1/3/8 {
    vlan-tagging;
  }
}
protocols {
  bgp {
    family inet-vpn {
      unicast {
        accept-local-nexthop;
      }
    }
  }
}

```

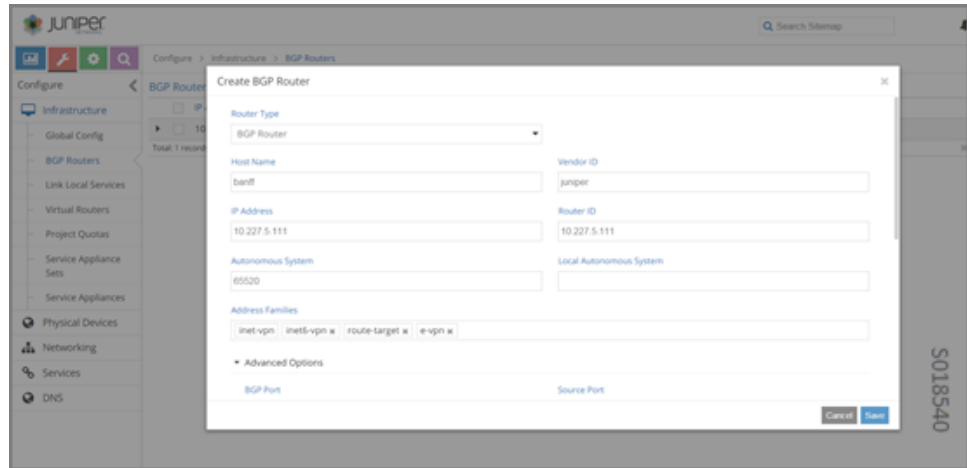
```
    }  
  }
```

If the MX is a service control gateway (SCG), the following configuration must also be present to support the service subscriptions:

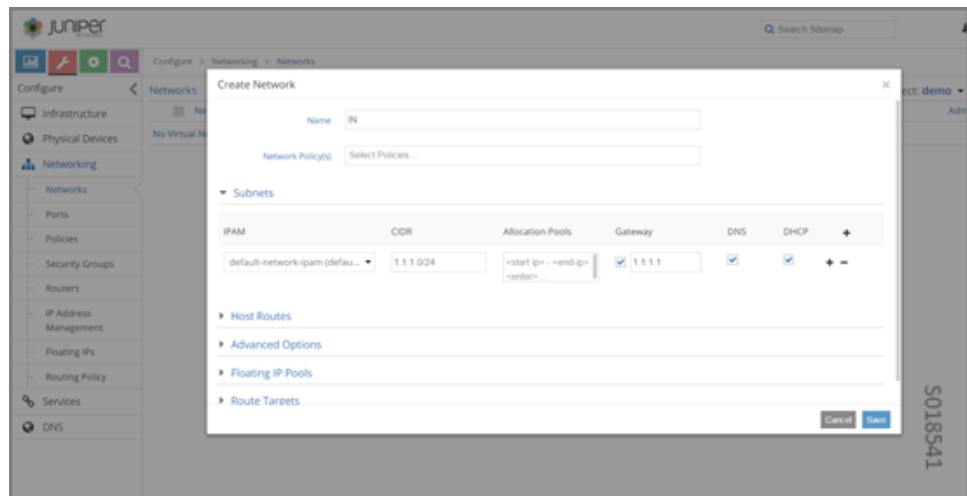
```
  firewall {  
    family inet {  
      filter skip_tdf_service {  
        term term1 {  
          then {  
            skip-services;  
            accept;  
          }  
        }  
      }  
    }  
  }  
}  
routing-instances {  
  <*-sc-entry-point> {  
    forwarding-options {  
      family inet {  
        filter {  
          input skip_tdf_service;  
        }  
      }  
    }  
  }  
}
```


Procedure: Adding a PNF to a Service Chain

1. At the Contrail UI, **Configure > Infrastructure > BGP Routers**, create a BGP router, with the Contrail controller as a peer, the address family you need, and a minimum configuration of the **route-target**, **inet**, and the **inet-vpn**. The following figure provides an example.

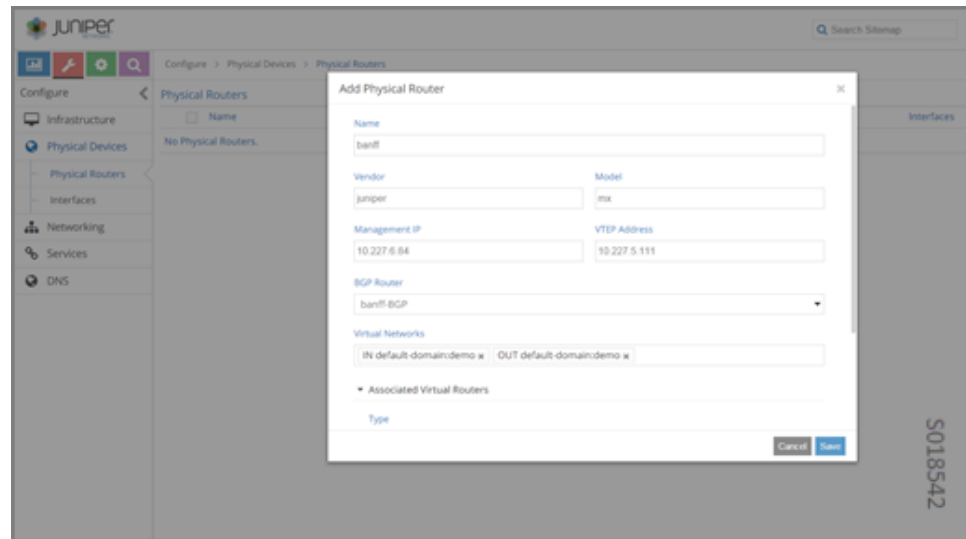


2. Create two virtual networks. Select **Configure > Networking > Networks** and create a network named **IN** and a network named **OUT**. The following figure provides an example.

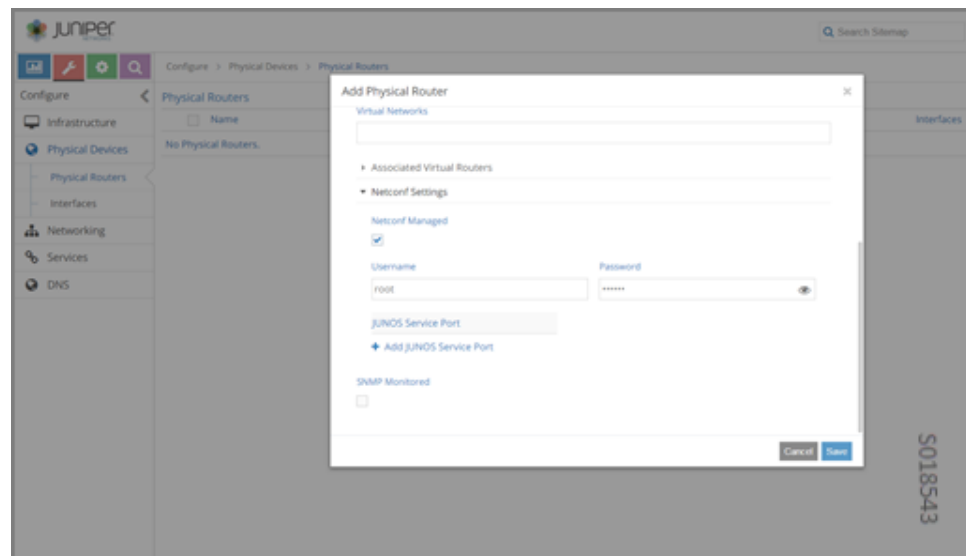


3. Create a physical router associated with the BGP router. Select **Configure > Physical Devices > Physical Routers** and create a physical router. The VTEP address of the physical router should be same as the BGP router's IP address. Associate the physical router with the BGP router created previously, and select for Virtual Networks the

networks created for this example (**IN** and **OUT**). The following figure provides an example.

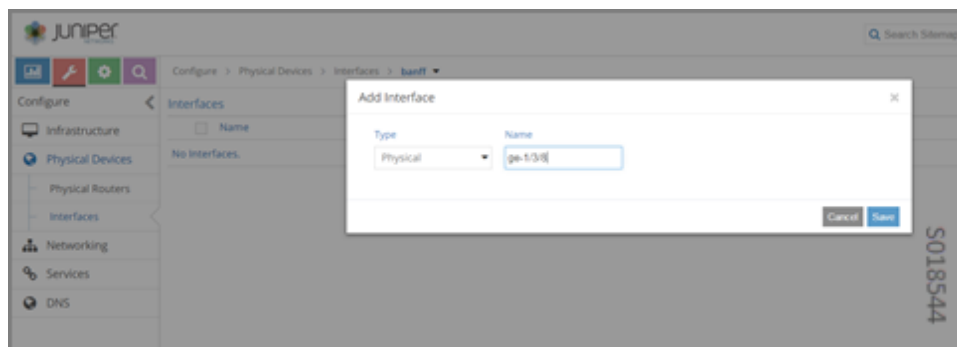


4. While still on the **Add Physical Router** window, use the slider to scroll down to the **Netconf Settings** section and add the appropriate NETCONF information for your system. The following figure provides an example.

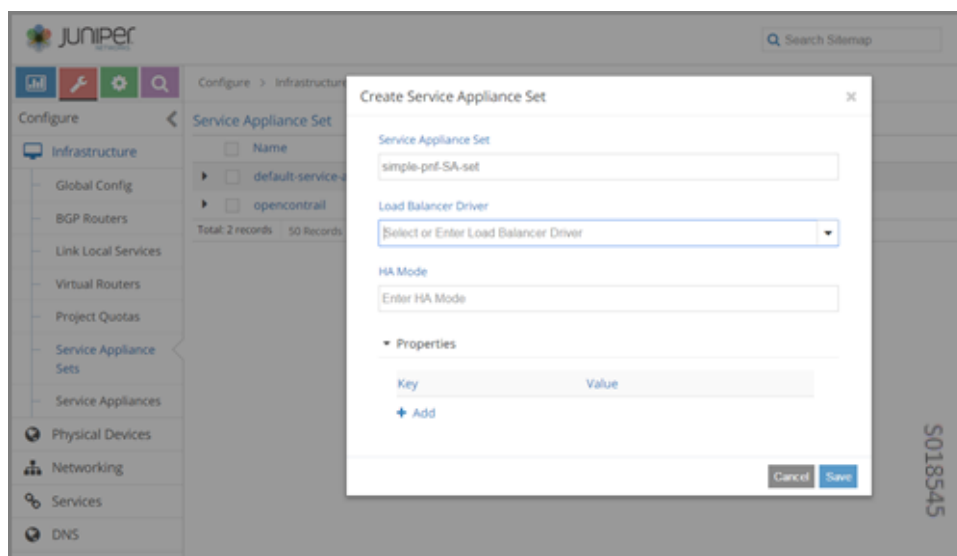


5. Add the physical interfaces that connect to the PNF device. Go to **Configure > Physical Devices > Interfaces** and select the PNF to get to the **Add Interfaces** window, where

you enter the name and type for each interface. The following figure provides an example.

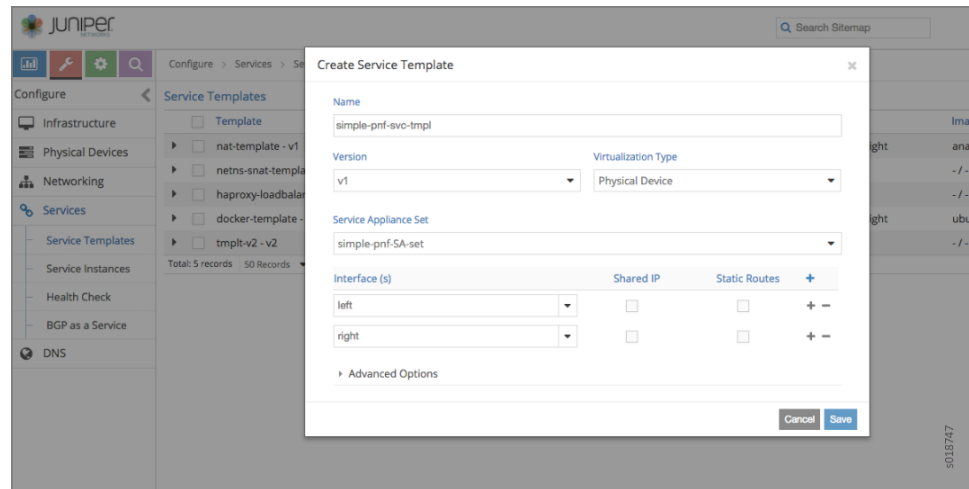


6. Add a service appliance set. Go to **Configure > Infrastructure > Service Appliance Sets** to get to the **Create Service Appliance Set** window, where you enter the name of the service appliance set. The following figure provides an example.

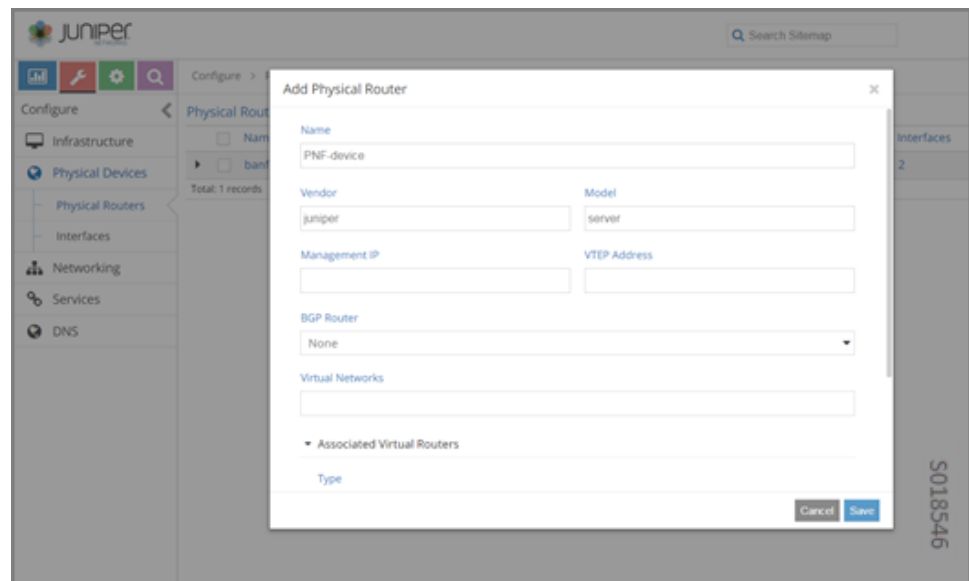


7. Configure a service template, **Configure > Services > Service Templates** and click the **Create** button on **Service Templates** to get to **Add Service Template**. Ensure that the

Virtualization Type is set to **Physical Device**, and that the template is associated to the service appliance set previously created. The following figure provides an example.

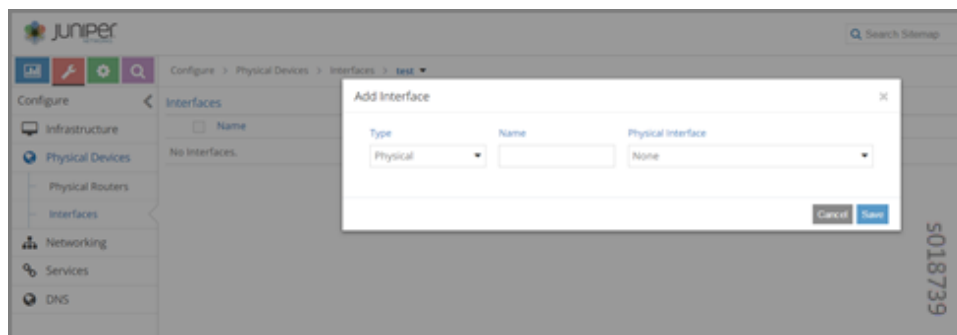


8. Add a physical router that represents the PNF device. Go to **Configure > Physical Devices > Physical Routers** to get to the **Add Physical Router** window, where you enter a name for the physical router. The following figure provides an example.

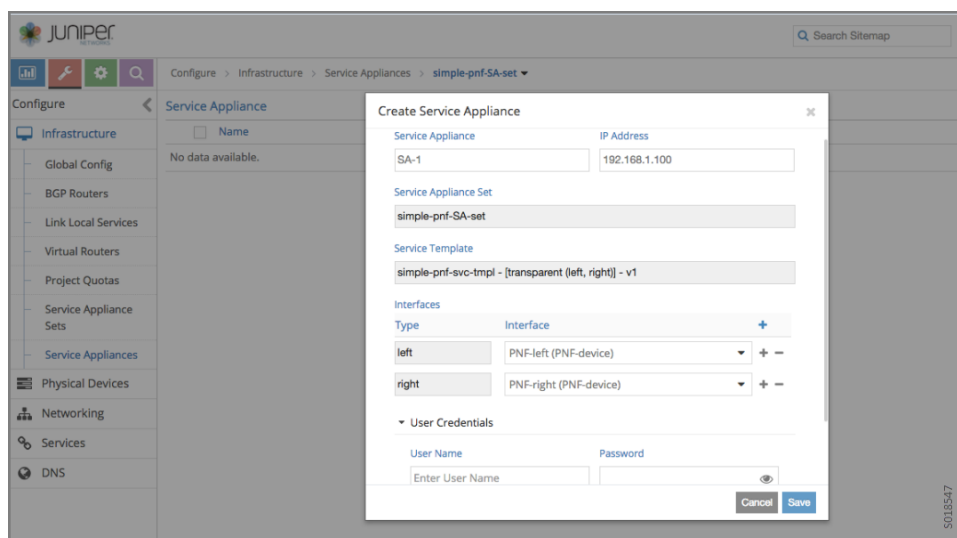


9. Create two interfaces for the PNF. The interfaces should connect to the interfaces already created in this example, and should connect in the manner illustrated in the

topology diagram. The interfaces for the other PR should be available from the selection field. The following figure provides an example.



10. Add a service appliance in the service appliance set. Go to **Configure > Infrastructure > Service Appliances** to get to the **Create Service Appliance** window, where you enter the name of the service appliance set and the IP address. Also add the left and right interfaces previously created. The following figure provides an example.



The remaining steps are the same as the steps to create a Contrail service chain, and are summarized in the following steps.

For more details about service chains, see:

- [Service Chaining on page 357](#)
- [Example: Creating a Transparent Service Chain on page 405](#)

11. Create a PNF service instance, go to **Configure > Services > Service Instances**, and click **Create**, then select the template to use and select the corresponding left, right, or management networks. When using a transparent service chain, the VN for the interfaces can be automatic.

12. Add a network policy to connect the virtual networks created for this example, go to **Configure > Networking > Policies**.
13. Associate the policy to both the left VN and the right VN (**IN** and **OUT** in this example). Navigate to **Configure > Networking > Network**.

**Related
Documentation**

- [Service Chaining on page 357](#)
- [Example: Creating a Transparent Service Chain on page 405](#)
- [Using Physical Network Functions in Contrail Service Chains on page 413](#)

CHAPTER 18

Configuring High Availability

- [Juniper OpenStack High Availability on page 423](#)
- [High Availability Support Options on page 430](#)
- [High Availability for Containerized Contrail on page 433](#)

Juniper OpenStack High Availability

- [Introduction on page 423](#)
- [Contrail High Availability on page 424](#)
- [OpenStack High Availability on page 424](#)
- [Supported Platforms on page 424](#)
- [Juniper OpenStack High Availability Architecture on page 424](#)
- [Juniper OpenStack Objectives on page 425](#)
- [Limitations on page 425](#)
- [Solution Components on page 426](#)
- [Virtual IP with Load Balancing on page 426](#)
- [Failure Handling on page 426](#)
- [Deployment on page 427](#)
- [Minimum Hardware Requirement on page 427](#)
- [Compute on page 427](#)
- [Network on page 428](#)
- [Installation on page 428](#)
- [Sample Server Manager-JSON on page 429](#)

Introduction

The Juniper Networks software-defined network (SDN) controller has two major components: OpenStack and Contrail. High availability (HA) of the controller requires that both OpenStack and Contrail are resistant to failures. Failures can range from a service instance failure, node failure, link failure, to all nodes down due to a power outage. The basic expectation from a highly available SDN controller is that when failures occur, already provisioned workloads continue to work as expected without any traffic drop, and the controller is available to perform operations on the cluster. Juniper Networks

OpenStack is a distribution from Juniper Networks that combines OpenStack and Contrail into one product.

Contrail High Availability

Contrail has high availability already built into various components, including support for the Active-Active model of high availability, which works by deploying the Contrail node component with an appropriate required level of redundancy.

The Contrail control node runs BGP and maintains adjacency with the vRouter module in the compute nodes. Additionally, every vRouter maintains a connection with all available control nodes.

Contrail uses Cassandra as the database. Cassandra inherently supports fault tolerance and replicates data across the nodes participating in the cluster.

A highly available deployment of Contrail, at minimum, requires at least:

- **two** control nodes
- **three** config nodes (including analytics and webui)
- **three** database nodes

OpenStack High Availability

High availability of OpenStack is supported by deploying the OpenStack controller nodes in a redundant manner on multiple nodes. Previous releases of Contrail supported only a single instance of the OpenStack controller, and multiple instances of OpenStack posed new problems that needed to be solved, including:

- State synchronization of stateful services (e.g. MySQL) across multiple instances.
- Load-balancing of requests across the multiple instances of services.

Supported Platforms

Juniper OpenStack Controller has tested high availability on the following platforms:

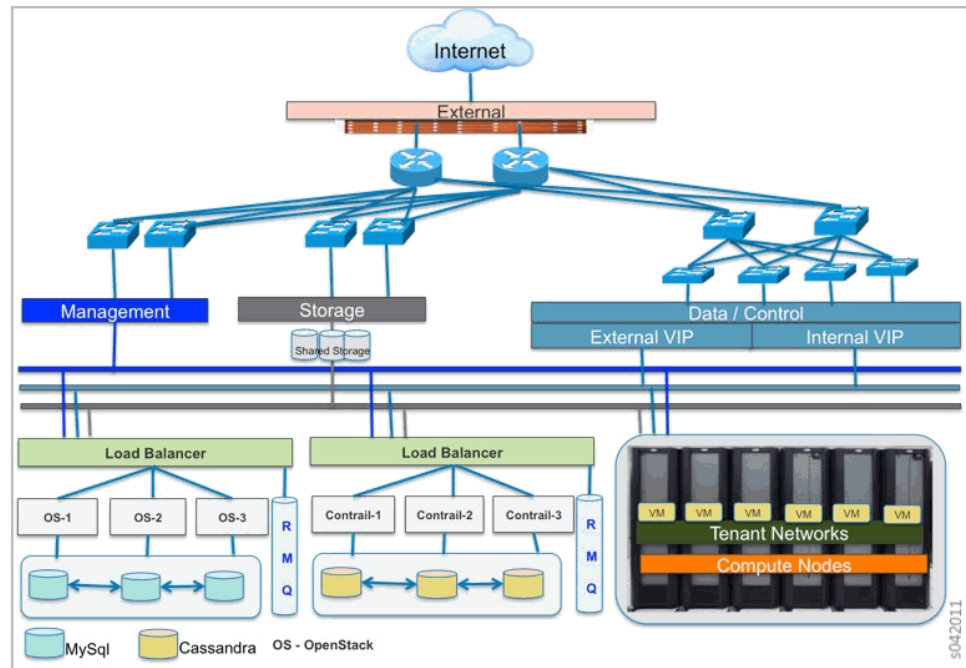
- Linux — Ubuntu 12.04 with kernel version 3.13.0-34
- Ubuntu Server 16.04 LTS (Xenial Xerus)

For a list of all operating system versions and the corresponding Linux or Ubuntu kernel versions supported by Contrail Release 4.0 on OpenStack Kilo, Liberty, Mitaka, Newton and Ocata releases, see [“Supported Platforms Contrail 4.0.x” on page 9](#).

Juniper OpenStack High Availability Architecture

A typical cloud infrastructure deployment consists of a pool of resources of compute, storage, and networking infrastructure, all managed by a cluster of controller nodes.

The following figure illustrates a high-level reference architecture of a high availability deployment using Juniper OpenStack deployed as a cluster of controller nodes.



Juniper OpenStack Objectives

The main objectives and requirements for Juniper OpenStack high availability are:

- 99.999% availability for tenant traffic.
- Anytime availability for cloud operations.
- Provide VIP-based access to the API and UI services.
- Load balance network operations across the cluster.
- Management and orchestration elasticity.
- Failure detection and recovery.

Limitations

The following are limitations of Juniper OpenStack high availability:

- Only one failure is supported.
- During failover, a REST API call may fail. The application or user must reattempt the call.
- Although zero packet drop is the objective, in a distributed system such as Contrail, a few packets may drop during ungraceful failures.
- Juniper OpenStack high availability is not tested with any third party load balancing solution other than HAProxy.

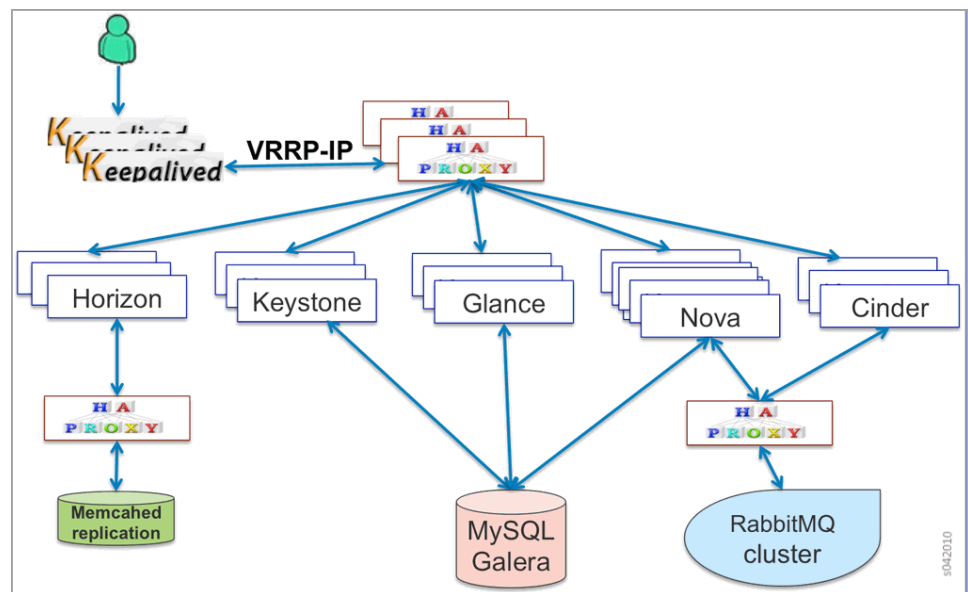
Solution Components

Juniper Openstack's high availability active-active model provides scale out of the infrastructure and orchestration services. The model makes it very easy to introduce new services in the controller and in the orchestration layer.

Virtual IP with Load Balancing

HAProxy is run on all nodes to load balance the connections across multiple instances of the services. To provide a Virtual IP (VIP), Keepalived (open source health check framework and hot standby protocol) runs and elects a master based on VRRP protocol. The VRRP master owns the VIP. If the master node fails, the VIP moves to a new master elected by VRRP.

The following figure shows OpenStack services provisioned to work with HAProxy and Keepalived, with HAProxy at the front of OpenStack services in a multiple operating system node deployment. The OpenStack database is deployed in clustered mode and uses Galera for replicating data across the cluster. RabbitMQ has clustering enabled as part of a multinode Contrail deployment. The RabbitMQ configuration is further tuned to support high availability.



Failure Handling

This section describes how various types of failures are handled, including:

- Service failures
- Node failures
- Networking failures

Service Failures

When an instance of a service fails, HAProxy detects the failure and load balances any subsequent requests across other active instances of the service. The supervisor process monitors for service failures and brings up the failed instances. As long as there is one instance of a service operational, the Juniper OpenStack controller continues to operate. This is true for both stateful and stateless services across Contrail and OpenStack.

Node Failures

The Juniper OpenStack controller supports single node failures involving both graceful shutdown or reboots and ungraceful power failures. When a node that is the VIP master fails, the VIP moves to the next active node, as it is elected to be the VRRP master. HAProxy on the new VIP master sprays the connections over to the active service instances as before, while the failed down node is brought back online. Stateful services (MySQL, Galera, Zookeeper, and so on) require a quorum to be maintained when a node fails. As long as a quorum is maintained, the controller cluster continues to work without problems. Data integrity is also inherently preserved by Galera, Rabbit, and other stateful components in use.

Network Failures

A connectivity break, especially in the control data network causes the controller cluster to partition into two. As long as the caveat of minimum number of nodes is maintained for one of the partitions, the controller cluster continues to work. Stateful services detect the partitioning and reorganize their cluster around the reachable nodes. Existing workloads continue to function and pass traffic and new workloads can be provisioned. When the connectivity is restored, the joining node becomes part of the working cluster and the system gets restored to its original state.

Deployment

Minimum Hardware Requirement

A minimum of 3 servers (physical or virtual machines) are required to deploy a highly available Juniper OpenStack Controller. In Active-Active mode, the controller cluster uses Quorum-based consistency management for guaranteeing transaction integrity across its distributed nodes. This translates to the requirement of deploying $2n+1$ nodes to tolerate n failures.

The Juniper OpenStack Controller offers a variety of deployment choices. Depending on the use case, the roles can be deployed either independently or in some combined manner. The type of deployment determines the sizing of the infrastructure. The numbers below present minimum requirements across compute, storage, and network.

Compute

- Quad core Intel(R) Xeon 2.5 Gz or higher
- 32 GB or higher RAM for the controller hosts (increases with number of hypervisors being supported)
- Minimum 1 TB disk, SSD, HDD

Network

A typical deployment separates control data traffic from the management traffic.

- Dual 10 GE that is bonded (using LAG 802.3ad) for redundant control data connection.
- Dual 1 GE bonded (using LAG 802.3 ad) for redundant management connection.
- Single 10G and 1G can be used if link redundancy is not desired.

The deployment needs virtual IP (VIP) addresses from the networks in which the NICs participate, external VIP on the management network and internal VIP on the control data network. External facing services are load balanced using the external VIP and the internal VIP is used for communication between other services.

Packaging

High availability support requires new components in the Contrail OpenStack deployment, which are packaged in **contrail-openstack-ha**, including HAProxy, Keepalived, Galera, and their requisite dependencies.

Installation

Installation is supported through Server Manager provisioning. The cluster JSON file has parameters specifying external and internal VIPs. If OpenStack and Contrail roles are co-located on the nodes, only one set of external and internal VIPs is needed.

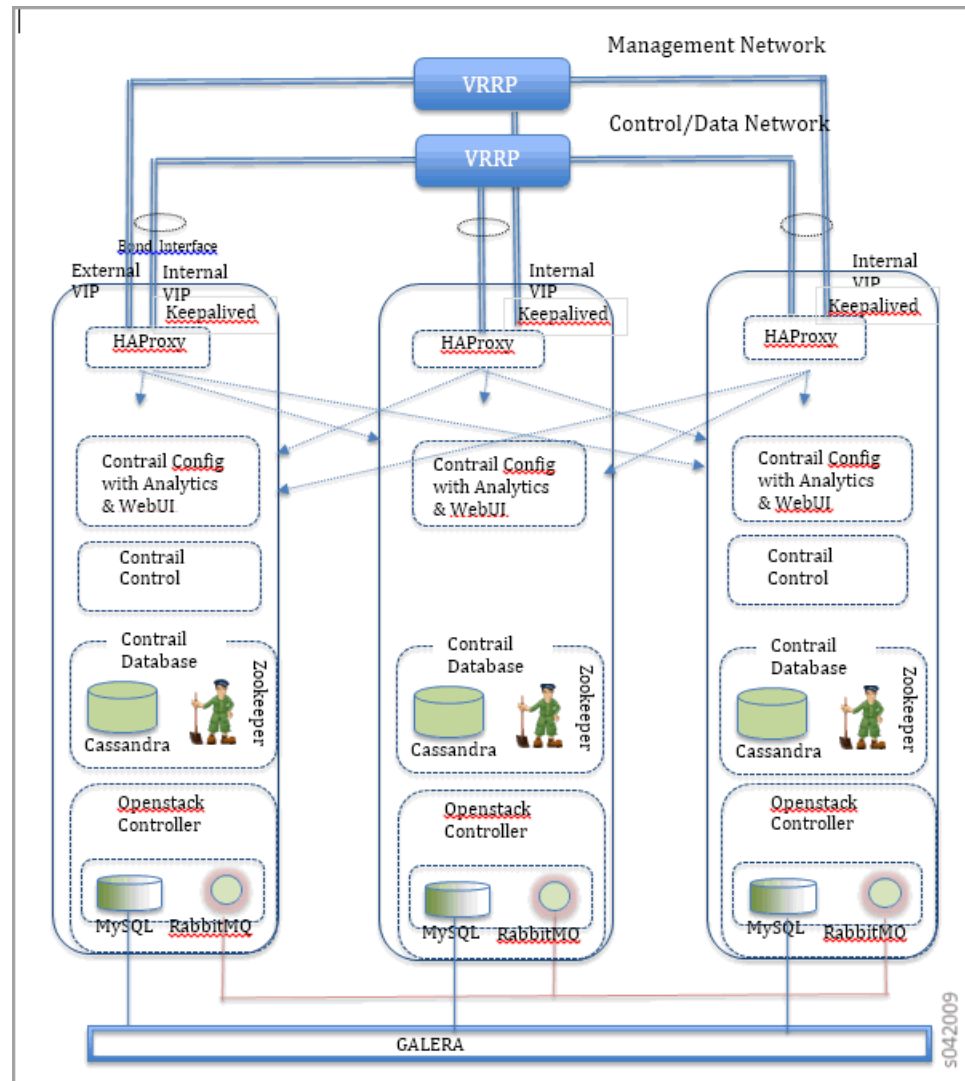
Install also supports separating OpenStack and Contrail roles on physically different servers. In this case, the external and internal VIPs specified are used for the OpenStack controller, and a separate set of VIPs, **contrail_external_vip** and **contrail_internal_vip**, are used for the Contrail controller nodes. It is also possible to specify separate RabbitMQs for OpenStack and Contrail controllers.

The following services are configured during high availability-enabled provisioning:

- Keepalived — Configures VRRP and VIP using **keepalived** package
- high availability proxy — Configured to load balance among services running on different nodes
- Galera — Openstack MySQL clustering to achieve high availability
- Glance — Support NFS server storage for glance images
- Keystone — Not supported for Server Manager provisioning

Starting with Contrail Release 4.0, provisioning scripts use VIPs instead of the physical IP of the node in all OpenStack and Contrail configuration files. The following figure

shows a typical three-node deployment, where Openstack and Contrail roles are co-located on three servers.



Sample Server Manager-JSON

When Server Manager is installed, configure an appropriate JSON file with the IP addresses, interface names, and password strings specific to your system. Select a sample JSON from the following and use Server Manager commands to create Contrail objects. See [Sample JSONs for a Multinode Cluster with High Availability](#):

- Related Documentation**
- [High Availability Support Options on page 430](#)
 - [Example: Adding New OpenStack or Contrail Roles to an Existing High Availability Cluster](#)

High Availability Support Options

This section describes how to set up Contrail options for high availability support.

- [Contrail High Availability Features on page 430](#)
- [Configuration Options for Enabling Contrail High Availability on page 430](#)
- [Supported Cluster Topologies for High Availability on page 431](#)
- [Deploying OpenStack and Contrail on the Same Highly Available Nodes on page 431](#)
- [Deploying OpenStack and Contrail on Different High Available Nodes on page 431](#)
- [Deploying Contrail Only on High Available Nodes on page 432](#)

Contrail High Availability Features

The Contrail OpenStack high availability design and implementation provides:

- A high availability active-active implementation for scale-out of the cloud operation and for flexibility to expand the controller nodes to service the compute fabric.
- Anytime availability of the cloud for operations, monitoring, and workload monitoring and management.
- Self-healing of the service and states.
- VIP-based access to the cloud operations API provides an easy way to introduce new controllers and an API to the cluster with zero downtime. Improved capital efficiencies compared with dedicated hardware implementations, by using nodes assigned to controllers and making them a federated node in the cluster.
- Operational load distribution across the nodes in the cluster.

For more details about high availability implementation in Contrail, see [“Juniper OpenStack High Availability” on page 423](#).

Configuration Options for Enabling Contrail High Availability

The following are options available to configure high availability within the Contrail configuration file (**testbed.py**).

Option	Description
internal_vip	The virtual IP of the OpenStack high availability nodes in the control data network. In a single interface setup, the internal_vip will be in the management data control network.
external_vip	The virtual IP of the OpenStack high availability nodes in the management network. In a single interface setup, the external_vip is not required.
contrail_internal_vip	The virtual IP of the Contrail high availability nodes in the control data network. In a single interface setup, the contrail_internal_vip will be in the management data control network.

Option	Description
<code>contrail_external_vip</code>	The virtual IP of the Contrail high availability nodes in the management network. In a single interface setup, the <code>contrail_external_vip</code> is not required.
<code>nfs_server</code>	The IP address of the NFS server that will be mounted to <code>/var/lib/glance/images</code> for the openstack node. The default is to <code>env.roledefs['compute'][0]</code> .
<code>nfs_glance_path</code>	The NFS server path to save images. The default is to <code>/var/tmp/glance-images/</code> .
<code>openstack_manage_amqp</code>	A flag to indicate the node on which <code>rabbitmq</code> is set up. True indicates <code>rabbitmq</code> is setup on OpenStack nodes. False indicates <code>rabbitmq</code> is set up on OpenStack nodes and the controller container.

Supported Cluster Topologies for High Availability

This section describes configurations for the cluster topologies supported, including:

- OpenStack and Contrail on the same highly available nodes
- OpenStack and Contrail on different highly available nodes
- Contrail only on highly available nodes

Deploying OpenStack and Contrail on the Same Highly Available Nodes

OpenStack and Contrail services can be deployed in the same set of highly available nodes by setting the `internal_vip` parameter in the cluster configuration.

Because the high available nodes are shared by both OpenStack and Contrail services, it is sufficient to specify only `internal_vip`. However, if the nodes have multiple interfaces with management and data control traffic separated by provisioning multiple interfaces, then the `external_vip` also needs to be set in the cluster configuration.

Example

```
env.ha = {
    'internal_vip' : 'an-ip-in-control-data-network',
    'external_vip' : 'an-ip-in-management-network',
}
```

Deploying OpenStack and Contrail on Different High Available Nodes

OpenStack and Contrail services can be deployed on different high available nodes by setting the `internal_vip` and the `contrail_internal_vip` parameter in the cluster configuration.

Because the OpenStack and Contrail services use different high available nodes, it is required to separately specify `internal_vip` for OpenStack high available nodes and `contrail_internal_vip` for Contrail high available nodes. If the nodes have multiple interfaces, with management and data control traffic separated by provisioning multiple interfaces, then the `external_vip` and `contrail_external_vip` options also must be set in the cluster configuration.

Example

```
env.ha = {  
    'internal_vip' : 'an-ip-in-control-data-network',  
    'external_vip' : 'an-ip-in-management-network',  
    'contrail_internal_vip' : 'another-ip-in-control-data-network',  
    'contrail_external_vip' : 'another-ip-in-management-network',  
}
```

By default, the **rabbitmq** cluster is configured on OpenStack nodes. To manage a separate **rabbitmq** cluster for Contrail services, set the **openstack_manage_amqp** to **false** in the cluster configuration. In this case, OpenStack services use the **rabbitmq** cluster on OpenStack nodes and Contrail services use **rabbitmq** cluster on controller containers.

Example:

```
"openstack":{  
    "openstack_manage_amqp": false  
}
```

Deploying Contrail Only on High Available Nodes

Contrail services can be deployed only on a set of high available nodes by setting the **contrail_internal_vip** parameter in the cluster configuration.

Because the high available nodes are used by only Contrail services, it is sufficient to specify only **contrail_internal_vip**. If the nodes have multiple interfaces with management and data control traffic are separated by provisioning multiple interfaces, the **contrail_external_vip** also needs to be set in the cluster configuration.

Example

```
env.ha = {  
    'contrail_internal_vip' : 'an-ip-in-control-data-network',  
    'contrail_external_vip' : 'an-ip-in-management-network',  
}
```

By default, the **rabbitmq** cluster is configured on OpenStack nodes. To manage a separate **rabbitmq** cluster for Contrail services, set the **openstack_manage_amqp** to **false** in the cluster configuration. In this case, OpenStack services use the **rabbitmq** cluster on OpenStack nodes and Contrail services use **rabbitmq** cluster on controller containers.

Example:

```
"openstack":{  
    "openstack_manage_amqp": false  
}
```

- Related Documentation**
- [Juniper OpenStack High Availability on page 423](#)
 - [Example: Adding New OpenStack or Contrail Roles to an Existing High Availability Cluster](#)

High Availability for Containerized Contrail

Starting with Contrail 4.0, some modules of Contrail have been grouped by function and packaged in Docker containers. This document describes the Contrail container subsystems that can be deployed in high availability mode.

- [Containers for High Availability on page 433](#)
- [How High Availability is Handled in Contrail Containers on page 433](#)

Containers for High Availability

The following Contrail container subsystems can be deployed in high availability mode:

- **contrail-lb**
- **contrail-controller**
- **contrail-analytics**
- **contrail-analyticsdb**

How High Availability is Handled in Contrail Containers

This section describes the mechanisms for accomplishing high availability in each of the Contrail containers.

- [contrail-lb on page 433](#)
- [High Availability for contrail-controller Container on page 433](#)
- [contrail-analytics on page 434](#)
- [contrail-analyticsdb on page 434](#)

contrail-lb

The Contrail load balancer container **contrail-lb** runs HAProxy, a de facto standard open source load balancer, and the BIRD protocol internet routing daemon.

- HAProxy is used to load balance across multiple instances of Contrail services.
- BIRD is used to deploy **contrail-lb** in high availability mode.

Although the **contrail-lb** containers are expected to be deployed in different hosts, they can also be deployed in the same hosts where **contrail-controller** containers are deployed.

For more information about BIRD, see [BIRD Babel protocol documentation](#)

High Availability for contrail-controller Container

The **contrail-controller** container runs the following services that can be scaled or clustered.

- **contrail-api**—Can be scaled up. It is load-balanced by HAProxy running in the **contrail-lb** containers. All clients can connect to the load-balancer IP to communicate with **contrail-lb**.
- **cassandra**—Clustered. The cassandra and client libraries have built-in load-balancing and failure detection mechanisms, and there is no need for cassandra to be behind HAProxy. However, multiple instances are clustered during deployment of the **contrail-controller** containers. All clients connect to the list of **contrail-controller** container IPs to communicate with cassandra.
- **zookeeper**—Clustered. The zookeeper and client libraries have a built-in high availability using leader-follower architecture, and there is no need for zookeeper to be behind HAProxy. However, multiple instances are clustered during deployment of the **contrail-controller** containers. All clients connect to the list of **contrail-controller** container IPs to communicate with zookeeper.
- **rabbitmq**—Clustered. The rabbitmq and client libraries can handle multiple rabbitmq instances. However, multiple instances are clustered and mirrored in queues during deployment of the contrail-controller containers. All clients connect to the list of contrail-controller container IPs to communicate with rabbitmq.



NOTE: Only an *odd* number of contrail-controllers are supported, due to a limitation with zookeeper for leader-follower election.

contrail-analytics

The contrail-analytics container runs the following services that can be scaled or clustered.

- **contrail-analytics-api**—Can be scaled up. It is load-balanced by HAProxy running in the contrail-lb containers. All clients connect to the load-balancer IP to communicate with contrail-analytics-api.

contrail-analyticsdb

The **contrail-analyticsdb** container runs the following services that can be scaled or clustered.

- **cassandra**—Clustered. The cassandra and client libraries have built-in load-balancing and failure detection mechanisms, and there is no need for cassandra to be behind HAProxy. However, multiple instances are clustered during deployment of the contrail-controller containers. All clients connect to the list of contrail-analyticsdb container IPs to communicate with cassandra.
- **kafka**—Clustered. Kafka uses a zookeeper cluster running in the contrail-controller containers. Multiple instances of kafka are clustered during deployment of contrail-analyticsdb containers. All clients connect to the list of the **contrail-analyticsdb** container IPs to communicate with kafka.

Related Documentation

- [Juniper OpenStack High Availability on page 423](#)

CHAPTER 19

Multitenancy Support

- [Configuring Multitenancy Support on page 435](#)
- [Quality of Service in Contrail on page 438](#)
- [Configuring Network QoS Parameters on page 446](#)
- [BGP as a Service on page 448](#)
- [BGP as a Service in Contrail Release 3.1 on page 451](#)

Configuring Multitenancy Support

The following sections describe enabling and viewing multitenancy support.

- [Multitenancy Permissions on page 435](#)
- [API Server on page 436](#)
- [API Library Keystone Integration on page 437](#)
- [Supporting Utilities on page 437](#)

Multitenancy Permissions

The multi tenancy feature of the API server enables multiple tenants to coexist on the system without interfering with each other. This is achieved by encoding ownership information and permissions with each resource, allowing fine-grained control over create, read, update, and delete (CRUD) operations on those resources.

The Contrail **api-server** enforces resources permissions in a manner similar to Unix files. Each resource has an owner and group. Permissions associated with owner, group, and "others" are:

R - reading resource

W - create/update resource

X - link (refer to) object

CRUD permission requirements for resources managed by **api-server** are as follows:

C - write on parent object

For example, to create a virtual network requires write permission on the project.

R - read on object (parent if a collection)

U - write on object

D - write on parent

ref(link) - execute on object

For example, on a virtual network using **network-ipam**, **network-ipam** should have X permissions for owner, group, or "others".

API Server

If multitenancy is enabled, **api-server** deploys keystone middleware in its pipeline. The keystone middleware architecture supports a common authentication protocol in use between OpenStack projects.

The keystone middleware works in conjunction with **api-server** to derive the user name and role for each incoming request. Once obtained, the user name and role are matched against resource ownership and permissions. If the ownership matches or the permissions allow access, access is granted.

For example, assume Tenant A has the following attributes:

- owner = Bob
- group = Staff
- permissions = 750

In this example, only Bob can create a virtual network in Tenant A. Other staff members can view the virtual networks in Tenant A. No others can create or view any virtual networks in Tenant A.

Clients can obtain an **auth_token** by posting credentials to the keystone admin API (**/v2.0/tokens**). The **VncApi** client library does this automatically. If an **auth_token** is present in an incoming request, **api-server** validates credentials derived from the token against object permissions. If an incoming request has an invalid or missing **auth_token**, a 401 error is returned.

Notes:

- Multitenancy is enabled by the flag **multi_tenancy** in **/etc/contrail/api-server.conf**
- If multitenancy is enabled, **memcaching** is automatically enabled, to improve token validation response time.

API Library Keystone Integration

VncApi has been updated to check for any 401 error that **api-server** returns as a result of a missing or invalid token. This forces **VncApi** to connect with the keystone middleware and fetch an **auth_token**. All subsequent requests to **api-server** include the **auth_token**.

Supporting Utilities

- **/opt/contrail/utils/chmod.py**— To change permissions and ownership (user or group membership) of a resource. Requires the resource type (for example, **virtual-network**) and the resource FQN (for example, **default-domain:default-project:default-virtual-network**).

Invoke **python /opt/contrail/utils/chmod.py -h** to see usage information

Example 1 - See current permissions:

```
[root@host]# python /opt/contrail/utils/chmod.py <ip address> project
default-domain:default-project
Type = project
Name = default-domain:default-project
API Server = <ip address>
Keystone credentials admin/<password>/admin
Obj uuid = 6765f112-938f-4251-b3a9-fbbdcc09db18
Obj perms = cloud-admin/cloud-admin-group 777

[root@host]# python /opt/contrail/utils/chmod.py <ip address> --owner foo
--group bar --perms 555 project default-domain:default-project
Type = project Name = default-domain:default-project
API Server = <ip address>
Owner = foo
Group = bar
Perms = 555
Keystone credentials admin/<password>/admin
Obj uuid = 6765f112-938f-4251-b3a9-fbbdcc09db18
Obj perms = cloud-admin/cloud-admin-group 777
New perms = foo/bar 555
```

- **/opt/contrail/utils/multi_tenancy.py** — Show if multitenancy is enabled or disabled. Also used to turn multitenancy on or off. Requires admin credentials.

Invoke **python /opt/contrail/utils/multi_tenancy.py -h** to see usage information

Example 1: View multitenancy status:

```
[root@host]# python /opt/contrail/utils/multi_tenancy.py <ip address>
API Server = <ip address>
Keystone credentials admin/<password>/admin

Multi Tenancy is enabled
```

Example 2: Turn multitenancy off:

```
[root@host]# python /opt/contrail/utils/multi_tenancy.py <ip address>--off
API Server = <ip address>
Keystone credentials admin/<password>/admin

Multi Tenancy is disabled
```

Quality of Service in Contrail

- [Overview: Quality of Service on page 438](#)
- [Contrail QoS Model on page 438](#)
- [QoS Configuration Parameters for Provisioning on page 439](#)
- [Queuing Implementation on page 441](#)
- [Contrail QoS Configuration Objects on page 441](#)
- [Example: Mapping Traffic to Forwarding Classes on page 443](#)
- [QoS Configuration Object Marking on the Packet on page 444](#)
- [Queuing on page 444](#)
- [Queue Selection in Datapath on page 445](#)
- [Parameters for QoS Scheduling Configuration on page 445](#)

Overview: Quality of Service

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

Contrail QoS Model

The Contrail QoS model has the following features:

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

Features of Fabric Interfaces

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

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

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

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

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

QoS Configuration Parameters for Provisioning

- [Testbed.py Parameters on page 439](#)
- [JSON Parameters on page 440](#)

Testbed.py Parameters

Testbed.py can be used for provisioning Contrail through Releases 3.x.x. Starting with Contrail 4.0, **testbed.py** can only be used if you are provisioning with SM-Lite. Use parameters in this section if you are using **testbed.py** for provisioning.

For QoS, the hardware queues (NIC queues) are mapped to logical queues in the agent, using the following keys:

- **hardware_q_id**—Identifier for the hardware queue.
- **logical_queue**— Defines the logical queues to map to each hardware queue.
- **default**—Defines the default hardware queue for QoS when set to True.

Options to define a default hardware queue:

- Set the queue as default, without any logical queue mapping.

```
{'hardware_q_id': '1', 'default': 'True'}
```

- Set the hardware queue as default with logical queue mapping.

```
{'hardware_q_id': '6', 'logical_queue': ['17-20'], 'default': 'True'}
```

```
env.qos = {host4: [ {'hardware_q_id': '3', 'logical_queue': ['1',
'6-10', '12-15']},
                    {'hardware_q_id': '5', 'logical_queue': ['2']},
                    {'hardware_q_id': '8', 'logical_queue': ['3-5']},
                    {'hardware_q_id': '1', 'default': 'True'}],
  host5: [ {'hardware_q_id': '2', 'logical_queue': ['1', '3-8',
'10-15']},
           {'hardware_q_id': '6', 'logical_queue': ['17-20'],
'default': 'True'}]
}
```

The following are the keys for defining QoS priority groups.

- `priority_id`—Priority group for QoS.
- `scheduling`—Defines the scheduling algorithm used for the priority group, strict or roundrobin (rr).
- `bandwidth`—Total hardware queue bandwidth used by priority group.

Bandwidth cannot be specified if strict scheduling is used for priority group, so set it to 0.

Example: QoS Priority Group

```
env.qos_niantic = {host4:[
    { 'priority_id': '1', 'scheduling': 'strict', 'bandwidth': '0'},
    { 'priority_id': '2', 'scheduling': 'rr', 'bandwidth': '20'},
    { 'priority_id': '3', 'scheduling': 'rr', 'bandwidth': '10'},
    host5:[
    { 'priority_id': '1', 'scheduling': 'strict', 'bandwidth': '0'},
    { 'priority_id': '1', 'scheduling': 'rr', 'bandwidth': '30'}
    ]
}
```

Related Documentation

- [Sample testbed.py for Provisioning Containers with SM-Lite](#)

JSON Parameters

If you are provisioning using Server Manager or SM-Lite, you can provision configuration parameters using JSON files.

Example: JSON Configuration for QoS

```
{
  "parameters" : {
    "provision": {
      "contrail_4": {
        "qos": {
          "10": {
            "default": true,
            "logical_queue": [
              "7"
            ]
          },
          "30": {
            "logical_queue": [
              "1",
              "6-10",
              "12-15"
            ]
          },
          "5": {
            "logical_queue": [
              "2"
            ]
          },
          "8": {
            "logical_queue": [
              "3-5"
            ]
          }
        }
      },
      "qos_niantic": {
        "1": {
          "bandwidth": "0",

```



```

        "scheduling": "strict"
    },
    "2": {
        "bandwidth": "20",
        "scheduling": "rr"
    },
    "3": {
        "bandwidth": "10",
        "scheduling": "rr"
    }
},

```

Queuing Implementation

Starting with Contrail 3.2, queuing is added. The vRouter provides the infrastructure to use queues supplied by the network interface, a method that is also called hardware queueing. Network interface cards (NICs) that implement hardware queueing have their own set of scheduling algorithms associated with the queues. The Contrail implementation is designed to work with most NICs, however, the method is tested only on an Intel-based 10G NIC, also called Niantic.

QoS Features by Release

QoS features are introduced in the following Contrail releases:

- 3.1—QoS configuration and forwarding classes
- 3.2—queuing
- Not planned—egress marking and queuing

Contrail QoS Configuration Objects

Contrail QoS configuration objects include the:

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

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

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

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

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

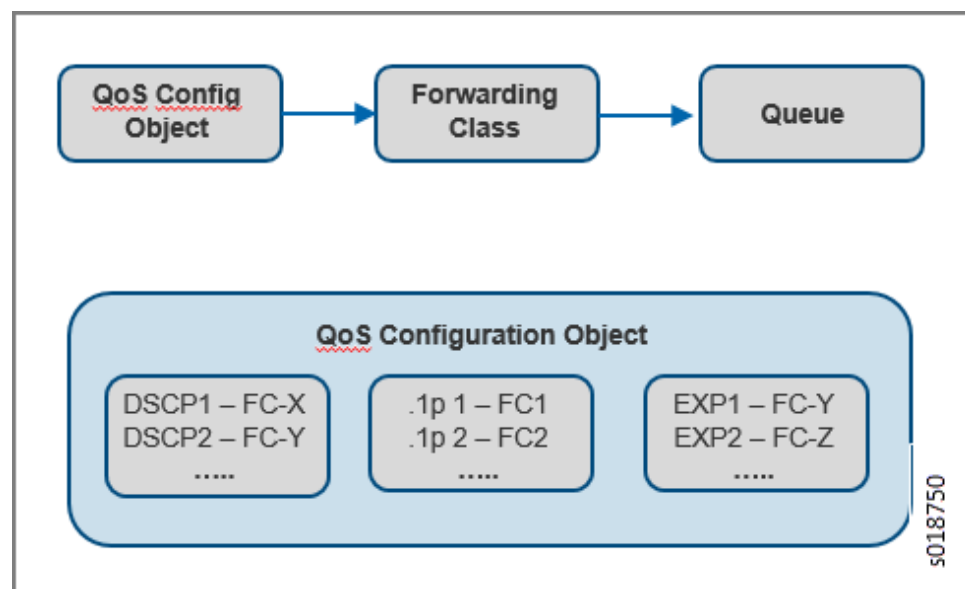
If the default forwarding class ID is not specified by the user, it defaults to the forwarding class with ID 0.

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

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

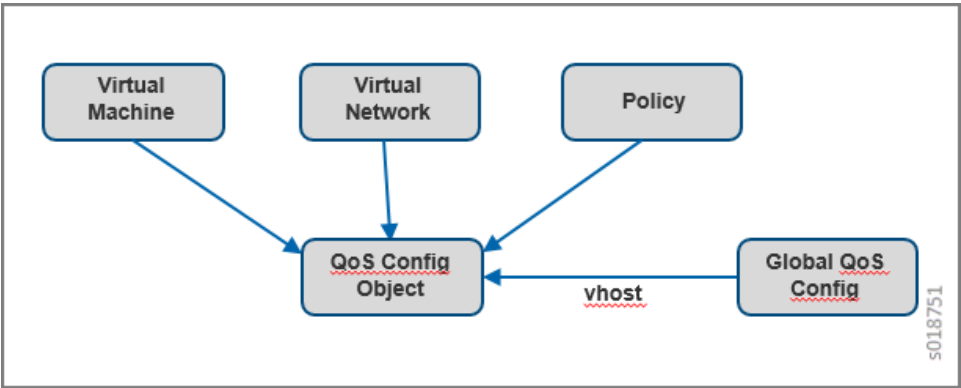
Figure 150 on page 442 shows the processing of QoS packets.

Figure 150: Processing of QoS Packets



A virtual machine interface, virtual network, and network policy can refer to the QoS configuration object. The QoS configuration object can be specified on the vhost so that underlay traffic can also be subjected to marking and queuing. See Figure 151 on page 443.

Figure 151: Referring to the QoS Object



Example: Mapping Traffic to Forwarding Classes

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

Table 48 on page 443 shows two forwarding class objects defined. FC1 marks the traffic with high priority values and queues it to Queue 0. FC2 marks the traffic as best effort and queues the traffic to Queue 1.

Table 48: Forwarding Class Mapping

Name	ID	DSCP	802.1p	MPLS EXP	Queue
FC1	1	10	7	7	0
FC2	2	38	0	0	1

In Table 49 on page 443, the QoS configuration object DSCP values of 10, 18, and 26 are mapped to a forwarding class with ID 1, which is forwarding class FC1. All other IP packets are mapped to the forwarding class with ID 2, which is FC2. All traffic with an 802.1p value of 6 or 7 are mapped to forwarding class FC1, and the remaining traffic is mapped to FC2.

Table 49: QoS Configuration Object Mapping

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

QoS Configuration Object Marking on the Packet

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

Traffic Originated by a Virtual Machine Interface

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

Traffic Destined to a Virtual Machine Interface

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

Traffic from a vhost Interface

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

Traffic from fabric interface

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

QoS Configuration Priority by Level

The QoS configuration can be specified at different levels.

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

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

Queuing

Contrail Release 3.2 adds QoS support for queuing.

This section provides an overview of the queuing features available starting with Contrail 3.2.

For more details about any of these topics, see:
<https://github.com/Juniper/contrail-controller/wiki/QoS> .

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

Queue Selection in Datapath

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

Hardware Queueing in Linux kernel based vRouter

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

To disable this mapping:

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

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

To verify:

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

Parameters for QoS Scheduling Configuration

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

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

Possible values for the scheduling algorithm include:

- strict
- rr (round-robin)

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

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

```
qos_niantic = {
  'compute1': [
```

```

    'bandwidth': '0'},
    '20'},
    '10'}
  ],
  'compute2' :[
    'bandwidth': '0'},
    '30'}
  ]
}
    { 'priority_id': '1', 'scheduling': 'strict',
    { 'priority_id': '2', 'scheduling': 'rr', 'bandwidth':
    { 'priority_id': '3', 'scheduling': 'rr', 'bandwidth':
    { 'priority_id': '1', 'scheduling': 'strict',
    { 'priority_id': '1', 'scheduling': 'rr', 'bandwidth':

```

- Related Documentation**
- [Configuring Network QoS Parameters on page 446](#)
 - <https://github.com/Juniper/contrail-controller/wiki/QoS>.

Configuring Network QoS Parameters

- [Overview on page 446](#)
- [QoS Configuration Examples on page 446](#)
- [Limitations on page 447](#)

Overview

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

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

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

QoS Configuration Examples

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

Example CLI for Nova flavor has the following format:

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

where:

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

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

QoS `vif` types include the following:

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

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

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

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

The following is an example of specified outbound parameters:

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

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

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

Limitations

- The stock `libvirt` does not support rate limiting of `ethernet` interface types. Consequently, settings like those in the example for the guest interface will not result in any `tc qdisc` settings for the corresponding tap device in the host. For more details, refer to issue [#1367095](#) in [Launchpad.net](#), where you can find patches and instructions to make `libvirt` work for network rate limiting of virtual machine interfaces.
- The `nova flavor-key rxtx_factor` takes a float as an input and acts as a scaling factor for receive (inbound) and transmit (outbound) throughputs. This key is only available to Neutron extensions (private extensions). The Contrail Neutron plugin doesn't

implement this private extension. Consequently, setting the **nova flavor-key rxtx_factor** will not have any effect on the QoS setting of the network interface(s) of any virtual machine created with that nova flavor.

- The outbound rate limits of a virtual machine interface are not strictly achieved. The outbound throughput of a virtual machine network interface is always less than the average outbound limit specified in the virtual machine's libvirt configuration file. The same behavior is also seen when using a Linux bridge.

See Also • [Quality of Service in Contrail on page 438](#)

BGP as a Service

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 on page 448](#)
- [BGPaaS Customer Use Cases on page 449](#)
- [Configuring BGPaaS on page 450](#)

Contrail BGPaaS Features

Using BGPaaS with Contrail 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.

Starting with Contrail Release 3.0, the following features have been added to BGPaaS:

- All BGPaaS sessions are configured to have bidirectional exchange of routes.
- 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.
- If multiple tenant VMs in the same virtual network have BGPaaS sessions and they use eBGP, standard loop prevention rules prevent routes advertised by one tenant VM from being advertised to other tenant VMs

A second BGP session for high availability can also be configured appropriately using one more BGP router object in the Contrail configuration and the peering session (from the VNF's point of view) to the DNS IP address (reserved by 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.

BGPaaS Customer Use Cases

This section provides example scenarios for implementing BGPaaS with Contrail.

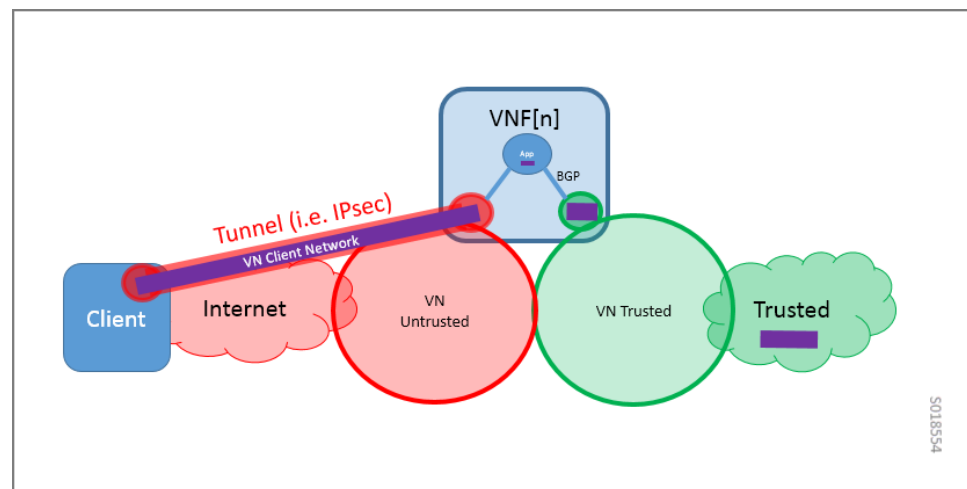
- [Dynamic Tunnel Insertion Within a Tenant Overlay on page 449](#)
- [Dynamic Network Reachability of Applications on page 449](#)
- [Liveness Detection for High Availability on page 450](#)

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.

The following figure is a generic example showing the need to insert a dynamic tunnel into a virtual network.



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

The following are methods for configuring BGPaaS:

- [Configuring BGPaaS Using VNC API on page 450](#)
- [Using the Contrail User Interface to Configure BGPaaS on page 451](#)

Configuring BGPaaS Using VNC API

To use VNC APIs to configure BGPaaS:

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 VMI.

```
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)
```

Deleting a BGPaaS Object

To delete a BGPaaS object:

```
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())
```

Using the Contrail User Interface to Configure BGPaaS

To configure BGPaaS within a tenant:

1. Within a tenant in Contrail, navigate to **Configure > Services > BGP as a Service**. Select the + icon to access the window **Create BGP as a Service**.

2. Enter the relevant information at the **Create BGP as a Service** window, including ASN, address family, and VMI identification.
3. Click **Save** to create the BGP object.

Related Documentation

- [BGP as a Service in Contrail Release 3.1 on page 451](#)

BGP as a Service in Contrail Release 3.1

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. For more information on BGP as a Service, see “[BGP as a Service](#)” on page 448.

The initial implementation of BGPaaS Version 1, supported in Contrail Release 3.0, allowed a tenant VM to establish BGP sessions to the default gateway and DNS server in the VM's subnet. A limitation of this implementation was that the tenant VM could advertise routes into the virtual network to which the VM belonged, however, the VM could not

receive any routes. The tenant VM was required to use a static default route, with the subnet's default gateway as the next hop.

Contrail Release 3.1 eliminates the previous limitation and 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 (VMI), not just a virtual machine (VM), which enables a tenant VM to have BGP sessions in multiple virtual networks, if required.

Starting with Contrail Release 3.1, the following features and properties have been added to BGPaaS:

- 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 its 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.

CHAPTER 20

Load Balancers

- [Using Load Balancers in Contrail on page 453](#)
- [Support for OpenStack LBaaS Version 2.0 APIs on page 465](#)
- [Configuring Load Balancing as a Service in Contrail on page 467](#)

Using Load Balancers in Contrail

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

- [Invoking LBaaS Drivers on page 453](#)
- [Using a Service Appliance Set as the LBaaS Provider on page 455](#)
- [Understanding the Load Balancer Agent on page 456](#)
- [F5 Networks Load Balancer Integration in Contrail on page 456](#)
- [Example: Creating a Load Balancer on page 460](#)
- [Using the Avi Networks Load Balancer for Contrail on page 461](#)

Invoking LBaaS Drivers

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

Supported load balancer drivers include:

- HAProxy
- A10 Networks
- F5 Networks
- Avi Networks

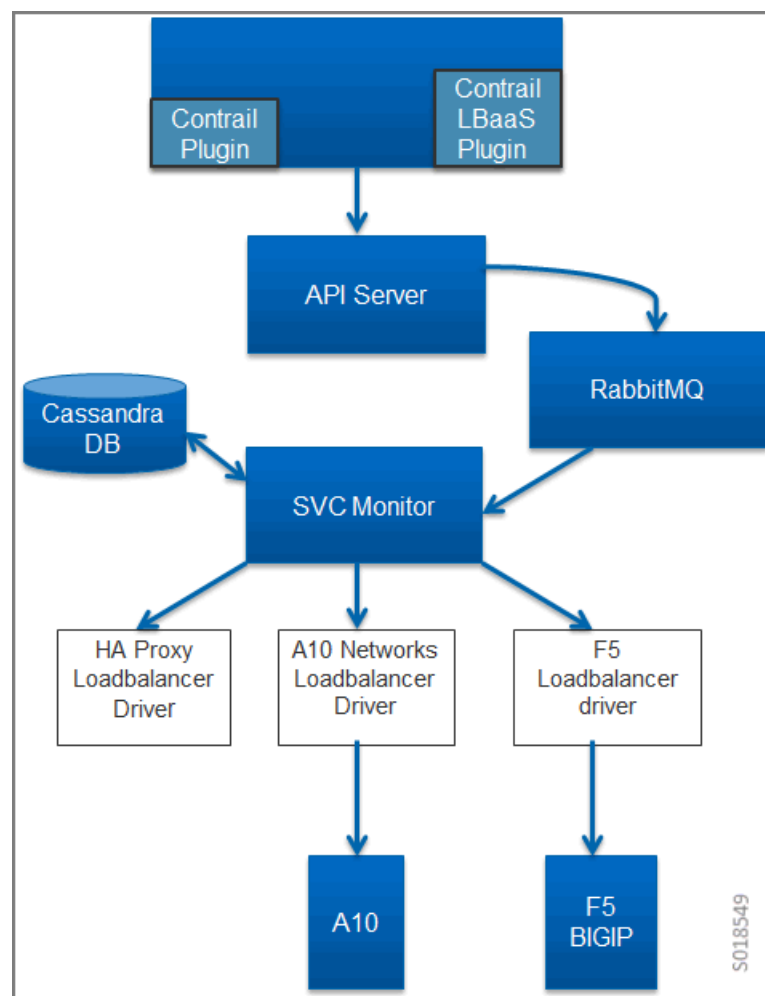
Starting with Contrail 3.0, the Neutron LBaaS plugin creates required configuration objects (such as pool, VIP, members, and monitor) in the Contrail API server, instead of within the Neutron plugin context, as in previous releases.

This method of configuration has the following benefits:

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

Figure 152 on page 454 provides an overview of the Contrail LBaaS components.

Figure 152: Contrail LBaaS Components



Using a Service Appliance Set as the LBaaS Provider

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

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

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

During initialization, the service monitor creates a default service appliance set with a default LBaaS provider, which uses an HAProxy-based load balancer. The service appliance set consists of individual service appliances for load balancing the traffic. The service appliances can be physical devices or virtual machines.

Sample Configuration: Service Appliance Set

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

```
{
  "service-appliance-set": {
    "fq_name": [
      "default-global-system-config",
      "f5"
    ],
    "service_appliance_driver":
      "svc_monitor.services.loadbalancer.drivers.f5.f5_driver.OpencontrailF5LoadbalancerDriver",

    "parent_type": "global-system-config",
    "service_appliance_set_properties": {
      "key_value_pair": [
        {
          "key": "sync_mode",
          "value": "replication"
        },
        {
          "key": "global_routed_mode",
          "value": "True"
        }
      ]
    },
    "name": "f5"
  }
}
```

Sample Configuration: Single Service Appliance

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

```
{
  "service-appliance": {
    "fq_name": [
```

```
        "default-global-system-config",
        "f5",
        "bigip"
    ],
    "parent_type": "service-appliance-set",
    "service_appliance_ip_address": "<ip address>",
    "service_appliance_user_credentials": {
        "username": "admin",
        "password": "<password>"
    },
    "name": "bigip"
}
```

Understanding the Load Balancer Agent

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

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

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

The load balancer agent module handles the following:

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

F5 Networks Load Balancer Integration in Contrail

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

- [F5 Load Balancer Global Routed Mode on page 457](#)
- [Initial Configuration on an F5 Device on page 459](#)
- [Initial Configuration on an MX Series Device Used as DC Gateway on page 459](#)

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

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

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

F5 Load Balancer Global Routed Mode

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

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

The following are features of the global routed mode.

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

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

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

Traffic Flow in Global Routed Mode

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

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

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

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

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

The MX device is preprovisioned with the following:

- VRF connected to pool network 2

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

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

The F5 device is preprovisioned with the following:

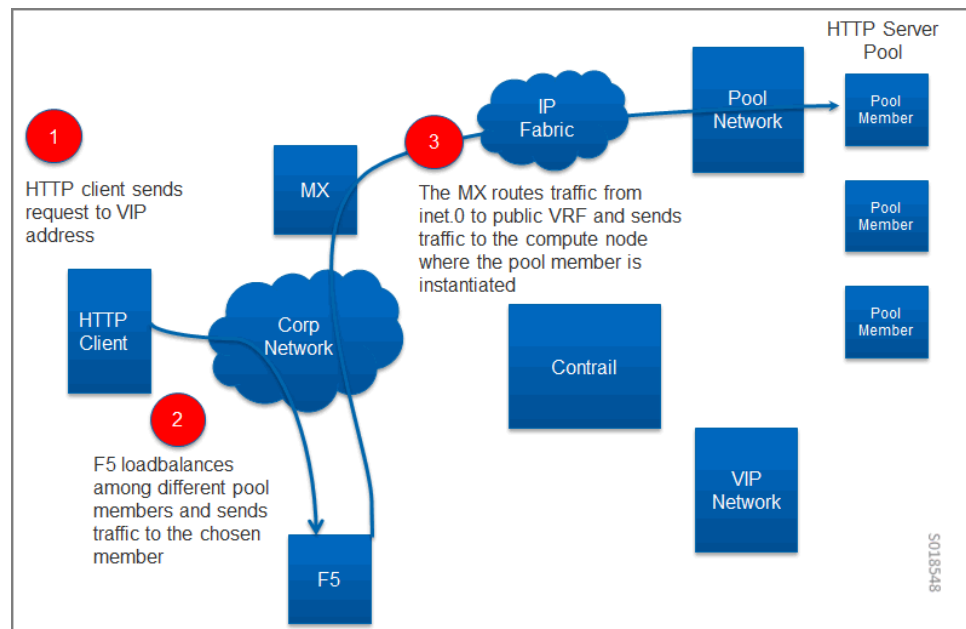
- publish route to attract VIP traffic
- pool network subnet route that points to the MX device

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

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

Figure 153 on page 458 shows the global routed traffic flow.

Figure 153: Global Routed Traffic Flow



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

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

Routing Traffic to Pool Members

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

Routing Reverse Traffic from Pool Members to the F5 Device

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

Initial Configuration on an F5 Device

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

Initial Configuration on an MX Series Device Used as DC Gateway

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

Configuration on MX Device for Each Pool Member

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

Example: Creating a Load Balancer

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

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

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

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

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

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

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

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

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

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

5. Create a VIP for the load balancer pool.

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

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

```
neutron lb-healthmonitor-create --delay 3 --type HTTP --max-retries 3 --timeout 3
neutron lb-healthmonitor-associate <nnnnn-nnnnn-nnnn-> web_service
```

Using the Avi Networks Load Balancer for Contrail

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

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



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

Installing the Avi LBaaS Neutron Driver

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

The following steps are performed on the Neutron server host.

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

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

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

```
$ vi /etc/neutron/neutron.conf
# if using mysql
connection = mysql+pymysql://neutron:cntrail123@127.0.0.1/neutron
```

3. Populate and upgrade the Neutron database schema.

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

4. Drop foreign key constraints.

```
# obtain current mysql token
$ cat /etc/contrail/mysql.token
fabe17d9dd5ae798f7ea
```

```
$ mysql -u root -p
Enter password: fabe17d9dd5ae798f7ea

mysql> use neutron;

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

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

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

- For a local installation:

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

<controller_ip|controller_vip>
--auser

--apass

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

--apass

--v2
```

- For a manual installation:

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

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

# LBaaS v2 driver
$ vi /etc/neutron/neutron.conf
#service_plugins =
```

```

neutron_plugin_contrail.plugins.opencontrail.loadbalancer.plugin.LoadBalancerPlugin
service_plugins =
neutron_lbaas.services.loadbalancer.plugin.LoadBalancerPluginv2
[service_providers]
service_provider =
LOADBALANCERV2:avi_adc_v2:neutron_lbaas.drivers.avi.driver.AviDriver

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

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

```

Installing the Avi LBaaS Contrail Driver

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

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

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

```

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

```

2. Install the Avi driver.

```

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

<controller_ip|controller_vip>
--auser

--apass

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

```

3. Set up the service appliance set.



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

```

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

```

4. To delete the service appliance set.

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

Configuring the Avi Controller

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

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

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

```
: > show cloud jcos
```

Field	Value
uuid	c1oud-104bb7e6-a9d2-4b34-a4c5-d94be659bb91
name	jcos
vtype	CLOUD_OPENSTACK
openstack_configuration	
username	admin
admin_tenant	demo
keystone_host	17x.xx.xx.50
mgmt_network_name	mgmtnw
privilege	WRITE_ACCESS
use_keystone_auth	True
region	RegionOne
hypervisor	KVM
tenant_se	True
import_keystone_tenants	True
anti_affinity	True
port_security	False
security_groups	True

		allowed_address_pairs	True
		free_floatingips	True
		img_format	OS_IMG_FMT_AUTO
		use_admin_url	True
		use_internal_endpoints	False
		config_drive	True
		insecure	True
		intf_sec_ips	False
		external_networks	False
		neutron_rbac	True
		nuage_port	8443
		contrail_endpoint	http://10.10.10.100:8082
		apic_mode	False
		dhcp_enabled	True
		mtu	1500 bytes
		prefer_static_routes	False
		enable_vip_static_routes	False
		license_type	LIC_CORES
		tenant_ref	admin
		+-----+-----+	

- Related Documentation**
- [Configuring Load Balancing as a Service in Contrail on page 467](#)
 - [Support for OpenStack LBaaS Version 2.0 APIs on page 465](#)

Support for OpenStack LBaaS Version 2.0 APIs

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

- [Platform Support on page 466](#)
- [Using OpenStack LBaaS Version 2.0 on page 466](#)
- [Support for Multiple Certificates per Listener on page 466](#)
- [Neutron Load-Balancer Creation on page 467](#)

Platform Support

Table 50 on page 466 shows which Contrail with OpenStack release combinations support which version of OpenStack LBaaS APIs.

Table 50: Contrail OpenStack Platform Support for LBaaS Versions

Contrail OpenStack Platform	LBaaS Support
Contrail-3.1-Liberty (and subsequent OS releases)	Only LBaaS v2 is supported.
Contrail-3.0-Liberty (and subsequent OS releases)	LBaaS v1 is default. LBaaS v2 is Beta.
<Contrail-any-release>-Kilo (and previous OS releases)	Only LBaaS v1 is supported.

Using OpenStack LBaaS Version 2.0

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

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

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

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

The object model has the following resources:

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

Support for Multiple Certificates per Listener

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

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

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

For more information about OpenStack Barbican, see:

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

Neutron Load-Balancer Creation

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

```
- neutron net-create private-net

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

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

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

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

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

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

Related Documentation

- OpenStack Barbican: <https://wiki.openstack.org/wiki/Barbican>
- OpenStack LBaaS v2.0 APIs: *LBaaS 2.0 (STABLE) (lbaas, loadbalancers, listeners, health_monitors, pools, members)*, at <http://developer.openstack.org/api-ref-networking-v2-ext.html>.
- Using Load Balancers in Contrail on page 453
- Configuring Load Balancing as a Service in Contrail on page 467

Configuring Load Balancing as a Service in Contrail

- Overview: Load Balancing as a Service on page 467
- Contrail LBaaS Implementation on page 469

Overview: Load Balancing as a Service

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

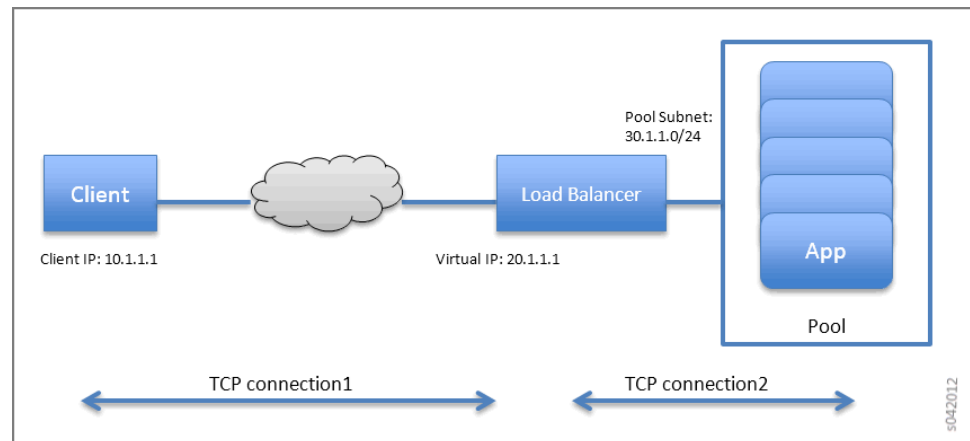
The LBaaS load balancer enables the creation of a pool of virtual machines serving applications, all front-ended by a virtual-ip. The LBaaS implementation has the following features:

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

In the following figure, the load balancer is launched with the virtual IP address 20.1.1.1. The backend pool of virtual machine applications (App Pool) is on the subnet 30.1.1.0/24. Each of the application virtual machines gets an IP address (virtual-ip) from the pool subnet. When a client connects to the **virtual-ip** for accessing the application, the load balancer proxies the TCP connection on its **virtual-ip**, then creates a new TCP connection to one of the virtual machines in the pool.

The pool member is selected using one of following methods:

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



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

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

Contrail LBaaS Implementation

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

Example: Configuring LBaaS

This feature is enabled on Contrail through Neutron API calls. The following is an example of creating a pool network and a VIP network. The VIP network is created in the public network and members are added in the pool network.

Creating a Load Balancer

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

1. Create a VIP network.

```
neutron net-create vipnet
```

```
neutron subnet-create --name vipsubnet vipnet 20.1.1.0/24
```

2. Create a pool network.

```
neutron net-create poolnet
```

```
neutron subnet-create --name poolsubnet poolnet 10.1.1.0/24
```

3. Create a pool for HTTP.

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

4. Add members to the pool.

```
neutron lb-member-create --address 10.1.1.2 --protocol-port 80 mypool
```

```
neutron lb-member-create --address 10.1.1.3 --protocol-port 80 mypool
```

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

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

Deleting a Load Balancer

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

1. Delete the VIP.

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

2. Delete members from the pool.

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

3. Delete the pool.

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

Managing Healthmonitor for Load Balancer

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

- Create a healthmonitor.

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

- Associate a healthmonitor to a pool.

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

- Disassociate a healthmonitor from a pool.

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

Configuring an SSL VIP with an HTTP Backend Pool

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

1. Copy an SSL certificate to all compute nodes.

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

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

```
# SSL certificate path haproxy
```

```
haproxy_ssl_cert_path=<certificate-path>
```

3. Restart `contrail-vrouter-agent`.

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

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

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

A Note on Installation

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

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

```
env.enable_lbaas=True
```

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

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

Limitations LBaaS currently has these limitations:

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

Related Documentation

- [Using Load Balancers in Contrail on page 453](#)
- [Support for OpenStack LBaaS Version 2.0 APIs on page 465](#)

CHAPTER 21

Optimizing Contrail

- [Route Target Filtering on page 473](#)
- [Source Network Address Translation \(SNAT\) on page 475](#)
- [Multiqueue Virtio Interfaces in Virtual Machines on page 478](#)
- [vRouter Command Line Utilities on page 479](#)

Route Target Filtering

- [Introduction on page 473](#)
- [Debugging and Troubleshooting Route Target Filtering on page 474](#)
- [RTF Limitations in Contrail 1.10 on page 475](#)

Introduction

BGP route target filtering (RTF) is a method for limiting the distribution of VPN routes to only those systems in the network for which the routes are necessary. If RTF is not active, the Contrail control node advertises all VPN routes to all of its VPN peers, which are either other control nodes or gateway routers such as an MX Series router. On the receiving side, the control node stores all VPN routes it receives from peers in the VPN table (for example, **bgp.l3vpn.0**). Any routes that do not include a route target extended community that is referenced by the local **vrf-import** policies are discarded by Junos.

The control node must send all route updates to its peers, even for unnecessary routes that are discarded. Continuous route updates are both CPU- and memory-intensive. The only routes that are necessary to advertise to gateway routers are those that belong to the virtual networks that are configured for public access. It is not necessary to advertise VM routes belonging to other virtual networks to gateway routers.

If a datacenter has more than two control nodes, the **vrouter-agent** only subscribes to two of the control nodes, indicated by the discovery service. When a VM is initially launched in a virtual network, it sends an XMPP subscribe request for the virtual network VRF and publishes the VM route to the connected control node. It is not necessary to advertise routes belonging to this type of VRF to control nodes that don't have the **vrouter-agent** subscribed in that VRF.

RTF is used to optimize the route distribution among control nodes and to the gateway routers to avoid unwanted route updates. If the BGP peer has not advertised or configured with RTF address family, then all routes belonging to the VPN table will be advertised.

RTF implementation in the control node does not support advertising and receiving of default route targets.

Constrained route distribution using route target reachability information is defined in RFC 4684, “*Constrained Route Distribution for Border Gateway Protocol/MultiProtocol Label Switching (BGP/MPLS) Internet Protocol (IP) Virtual Private Networks (VPNs)*”.

Debugging and Troubleshooting Route Target Filtering

Use the tips in this section to troubleshoot issues with RTF. Use various http introspect commands to reveal details about BGP neighbors for RTF. The following is a sample portion of an http introspect page.

When you access an introspect page, only the first panel of detail columns appears. Use a scroll bar or arrow keys to reveal more columns to the right, and vice versa.

BgpNeighborListResp						
neighbors						
peer	peer_address	deleted	peer_asn	local_address	local_asn	encoding
nodec13	10.204.216.70	false	0	10.204.216.70	0	XMPP
more						

- Use the following http introspect URL to display the details of each peer:

http://(your_node_name):8083/Snh_BgpNeighborReq

For BGP peers, verify the configured and negotiated capability and the BGP table registration.

For XMPP peers, look at the **routing_instances** column to get details about the VRF to which the displayed **vrouter-agent** has subscribed and to see the import **rtargets** of the VRFs.

- Use the following http introspect URL to dump the **bgp.rtarget.0** table to display the **RTargetRoutes**:

http://(your_node_name):8083/Snh_ShowRouteReq?x=bgp.rtarget.0

- Use the following http introspect URL to dump the details for each of the route targets configured on the control node:

http://(your_node_name):8083/Snh_ShowRtGroupReq?

For any given route target, this introspect displays the BGP table that imports and exports the route, the BGP peers that have shown interest in this route, and all dependent routes (when this route target has the extended community BGP attribute).

RTF Limitations in Contrail 1.10

The following are RTF limitations in Contrail 1.10.

- The control node does not support advertising a default route target, which is an **rtarget** route with **target:0:0** or **0/0** as the prefix. This type of **rtarget** route enables a BGP peer to receive all VPN routes without **rtarget** filtering.
- The control node does not support receiving a default route target. If **rtarget** routes with a default **rtarget** prefix are received, they are silently ignored.
- A **keep all** configuration, typical for BGP peering for a control node on an MX Series router, does not have impact, because all VPN routes with an extended community route target, for which the MX has advertised the **rtarget** route, are sent to the MX. An example of this type of typical configuration is the following:

```
set protocols bgp group contrail-control-nodes type internal
set protocols bgp group contrail-control-nodes local-address 10.204.216.253

set protocols bgp group contrail-control-nodes keep all
set protocols bgp group contrail-control-nodes family inet-vpn unicast
set protocols bgp group contrail-control-nodes family route-target
set protocols bgp group contrail-control-nodes neighbor 10.204.216.16
```

Source Network Address Translation (SNAT)

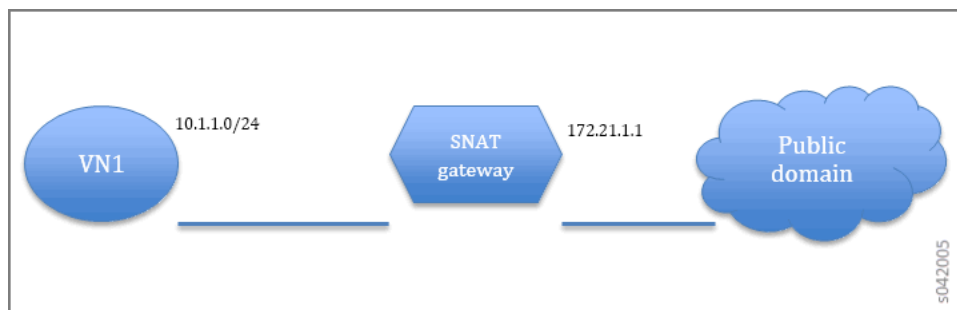
- [Overview on page 475](#)
- [Neutron APIs for Routers on page 476](#)
- [Network Namespace on page 477](#)
- [Using Web UI to Configure Routers with SNAT on page 477](#)

Overview

Source Network Address Translation (source-nat or SNAT) allows traffic from a private network to go out to the internet. Virtual machines launched on a private network can get to the internet by going through a gateway capable of performing SNAT. The gateway has one arm on the public network and as part of SNAT, it replaces the source IP of the originating packet with its own public side IP. As part of SNAT, the source port is also updated so that multiple VMs can reach the public network through a single gateway public IP.

The following diagram shows a virtual network with the private subnet of 10.1.1.0/24. The default route for the virtual network points to the SNAT gateway. The gateway replaces the source-ip from 10.1.1.0/24 and uses its public address 172.21.1.1 for outgoing packets. To maintain unique NAT sessions the source port of the traffic also needs to be replaced.

Figure 154: Virtual Network With a Private Subnet



Neutron APIs for Routers

OpenStack supports SNAT gateway implementation through its Neutron APIs for routers. The SNAT flag can be enabled or disabled on the external gateway of the router. The default is True (enabled).

The OpenContrail plugin supports the Neutron APIs for routers and creates the relevant service-template and service-instance objects in the API server. The service scheduler in OpenContrail instantiates the gateway on a randomly-selected virtual router. OpenContrail uses network namespace to support this feature.

Example Configuration: SNAT for Contrail

The SNAT feature is enabled on OpenContrail through Neutron API calls.

The following configuration example shows how to create a test network and a public network, allowing the test network to reach the public domain through the SNAT gateway.

1. Create the public network and set the router external flag.

```
neutron net-create public
neutron subnet-create public 172.21.1.0/24
neutron net-update public -- --router:external=True
```

2. Create the test network.

```
neutron net-create test
neutron subnet-create --name test-subnet test 10.1.1.0/24
```

3. Create the router with one interface in test.

```
neutron router-create r1
neutron router-interface-add r1 test-subnet
```

4. Set the external gateway for the router.

```
neutron router-gateway-set r1 public
```

Network Namespace

Setting the external gateway is the trigger for OpenContrail to set up the Linux network namespace for SNAT.

The network namespace can be cleared by issuing the following Neutron command:

```
neutron router-gateway-clear r1
```

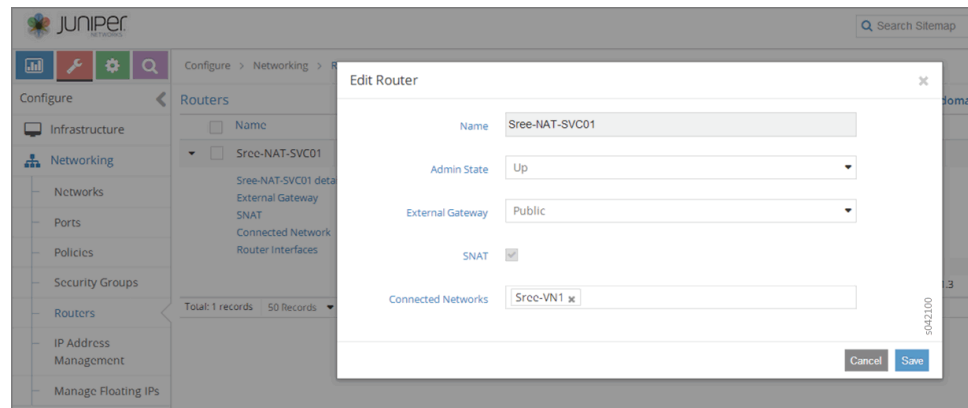
Using Web UI to Configure Routers with SNAT

You can use the Contrail user interface to configure routers for SNAT and to check the SNAT status of routers.

To enable SNAT for a router, go to **Configure > Networking > Routers**. In the list of routers, select the router for which SNAT should be enabled. Click the Edit cog to reveal the **Edit Routers** window. Click the check box for SNAT to enable SNAT on the router.

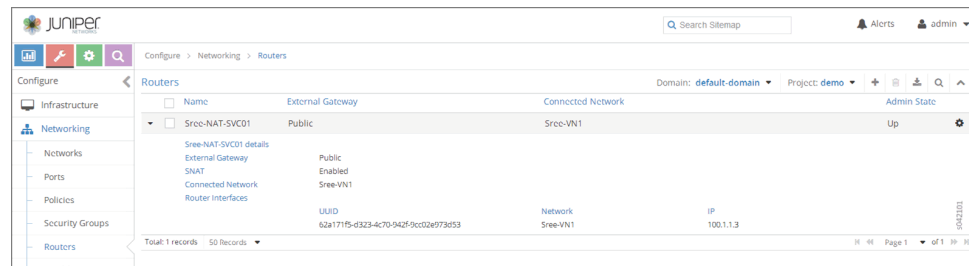
The following shows a router for which SNAT has been **Enabled**.

Figure 155: Edit Router Window to Enable SNAT



When a router has been **Enabled** for SNAT, the configuration can be seen by selecting **Configure > Networking > Routers**. In the list of routers, click open the router of interest. In the list of features for that router, the status of SNAT is listed. The following shows a router that has been opened in the list. The status of the router shows that SNAT is **Enabled**.

Figure 156: Router Status for SNAT



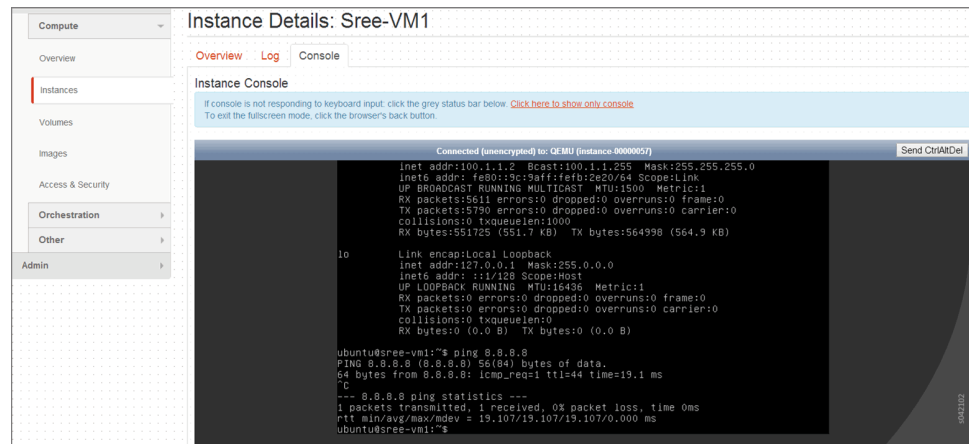
Name	External Gateway	Connected Network	Admin State
Sree-NAT-SVC01	Public	Sree-VN1	Up

Sree-NAT-SVC01 details
 External Gateway: Public
 SNAT: Enabled
 Connected Network: Sree-VN1
 Router Interfaces: Sree-VN1

UUID: 62a171f5-d323-4c70-942f-9cc02e973d53
 Network: Sree-VN1
 IP: 100.1.1.3

You can view the real time status of a router with SNAT by viewing the instance console, as in the following.

Figure 157: Instance Details Window



Instance Details: Sree-VM1

Overview Log Console

Instance Console

If console is not responding to keyboard input, click the grey status bar below. [Click here to show only console](#)
 To exit the fullscreen mode, click the browser's back button.

```

Connected [mynetwork] to: OEMU [instance-00000053]
Send Ctrl+Del

inet addr:100.1.1.2 Bcast:100.1.1.255 Mask:255.255.0
inet6 addr: fe80:19c:9aff:fe20:64 Scope:Link
UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
RX packets:5511 errors:0 dropped:0 overruns:0 frame:0
TX packets:5790 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:551725 (551.7 KB) TX bytes:564998 (564.9 KB)

Link encap:Local Loopback
inet addr:127.0.0.1 Mask:255.0.0.0
inet6 addr: ::1/128 Scope:Host
UP LOOPBACK RUNNING MTU:16436 Metric:1
RX packets:0 errors:0 dropped:0 overruns:0 frame:0
TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:0
RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)

ubuntu@sree-vm1:~$ ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data:
64 bytes from 8.8.8.8: icmp_seq=1 ttl=44 time=19.1 ms
1
--- 8.8.8.8 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 15.107/19.107/19.107/0.000 ms
ubuntu@sree-vm1:~$
  
```

Multiqueue Virtio Interfaces in Virtual Machines

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

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

- [Multiqueue Virtio Overview on page 478](#)
- [Requirements and Setup for Multiqueue Virtio Interfaces on page 478](#)

Multiqueue Virtio Overview

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

Requirements and Setup for Multiqueue Virtio Interfaces

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

- The OpenStack version must be Liberty or greater.

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

Setting Virtual Machine Metadata for Multiple Queues

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

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

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

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

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



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

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

vRouter Command Line Utilities

- [Overview on page 479](#)
- [vif Command on page 480](#)
- [flow Command on page 482](#)
- [vrfstats Command on page 484](#)
- [rt Command on page 484](#)
- [dropstats Command on page 485](#)
- [mpls Command on page 489](#)
- [mirror Command on page 491](#)
- [vxlan Command on page 492](#)
- [nh Command on page 493](#)

Overview

This section describes the shell prompt utilities available for examining the state of the vrouter kernel module in Contrail.

The most useful commands for inspecting the Contrail vrouter module are summarized in the following table.

Command	Description
vif	Inspect vrouter interfaces associated with the vrouter module.
flow	Display active flows in a system.
vrfstats	Display next hop statistics for a particular VRF.
rt	Display routes in a VRF.
dropstats	Inspect packet drop counters in the vrouter.
mpls	Display the input label map programmed into the vrouter.
mirror	Display the mirror table entries.
vxlan	Display the vxlan table entries.
nh	Display the next hops that the vrouter knows.
--help	Display all command options available for the current command.

The following sections describe each of the vrouter utilities in detail.

vif Command

The vrouter requires vrouter interfaces (**vif**) to forward traffic. Use the **vif** command to see the interfaces that are known by the vrouter.



NOTE: Having interfaces only in the OS (Linux) is not sufficient for forwarding. The relevant interfaces must be added to vrouter. Typically, the set up of interfaces is handled by components like nova-compute or vrouter agent.

Example: vif --list

```
# vif --list
vif0/0 OS: pkt0
    Type:Agent HWaddr:00:00:5e:00:01:00 IPaddr:0
    Vrf:65535 Flags:L3 MTU:1514 Ref:2
    RX packets:6591 bytes:648577 errors:0
    TX packets:12150 bytes:1974451 errors:0
vif0/1 OS: vhost0
    Type:Host HWaddr:00:25:90:c3:08:68 IPaddr:0
    Vrf:0 Flags:L3 MTU:1514 Ref:3
    RX packets:3446598 bytes:4478599344 errors:0
    TX packets:851770 bytes:1337017154 errors:0
vif0/2 OS: p1p0p0 (Speed 1000, Duplex 1)
    Type:Physical HWaddr:00:25:90:c3:08:68 IPaddr:0
```



```

Vrf:0 Flags:L3 MTU:1514 Ref:22
RX packets:1643238 bytes:1391655366 errors:2812
TX packets:3523278 bytes:6806058059 errors:0
vif0/18 OS: tap3214fc7e-88
Type:Virtual HWaddr:00:00:5e:00:01:00 IPAddr:0
Vrf:13 Flags:PL3L2 MTU:9160 Ref:6
RX packets:60 bytes:4873 errors:0
TX packets:21 bytes:2158 errors:0

```

Table 51: vif Fields

vif Output Field	Description
vif0/X	The vrouter assigned name, where 0 is the router id and X is the index allocated to the interface within the vrouter.
OS: pkt0	The pkt0 (in this case) is the name of the actual OS (Linux) visible interface name. For physical interfaces, the speed and the duplex settings are also displayed.
Type:xxxxx	<p>Type:Virtual HWaddr:00:00:5e:00:01:00 IPAddr:0</p> <p>The type of interface and its IP address, as defined by vrouter. The values can be different from what is seen in the OS. Types defined by vrouter include:</p> <ul style="list-style-type: none"> • Virtual – Interface of a virtual machine (VM). • Physical – Physical interface (NIC) in the system. • Host – An interface toward the host. • Agent – An interface used to trap packets to the vrouter agent when decisions need to be made for the forwarding path.
Vrf:xxxxx	<p>Vrf:65535 Flags:L3 MTU:1514 Ref:2</p> <p>The identifier of the vrf to which the interface is assigned, the flags set on the interface, the MTU as understood by vrouter, and a reference count of how many individual entities actually hold reference to the interface (mainly of debugging value).</p> <p>Flag options identify that the following are enabled for the interface:</p> <ul style="list-style-type: none"> • P - Policy • L3 - Layer 3 forwarding • L2 - Layer 2 bridging • X - Cross connect mode, only set on physical and host interfaces, indicating that packets are moved between physical and host directly, with minimal intervention by vrouter. Typically set when the agent is not alive or not in good shape. • Mt - Mirroring transmit direction • Mr - Mirroring receive direction • Tc - Checksum offload on the transmit side. Valid only on the physical interface.
Rx	<p>RX packets:60 bytes:4873 errors:0</p> <p>Packets received by vrouter from this interface.</p>
Tx	<p>TX packets:21 bytes:2158 errors:0</p> <p>Packets transmitted out by vrouter on this interface.</p>

vif Options Use **vif --help** to display all options available for the vif command. Following is a brief description of each option.



NOTE: It is not recommended to use the following options unless you are very experienced with the system utilities.

```
# vif --help
Usage: vif [--create <intf_name> --mac <mac>]
          [--add <intf_name> --mac <mac> --vrf <vrf>
          --type [vhost|agent|physical|virtual][--policy, --mode
<mode:x>]]
          [--delete <intf_id>]
          [--get <intf_id>][--kernel]
          [--set <intf_id> --vlan <vlan_id> --vrf <vrf_id>]
          [--list]
          [--help]
```

Option	Description
--create	Creates a 'Host' interface with name <intf_name> and mac <mac> on the host kernel. The 'vhost0' interface that you see on Linux is a typical example of invocation of this command.
--add	Adds the existing interfaces in the host OS to vrouter, with type and flag options.
--delete	Deletes the interface from vrouter. The <intf_id> is the vrouter interface id as given by vif0/X , where X is the iid
--get	Displays a specific interface. The <intf_id> is the vrouter interface id, unless the command is appended by the '--kernel' option, in which case the ID can be the kernel ID.
--set	Set working parameters of an interface. The only ones supported are the vlan id and the vrf . The vlan id as understood by vrouter differs from what one typically expects, and is relevant as of now only for interfaces of service instances.
--list	Display all of the interfaces of which the vrouter is aware.
--help	Display all options available for the current command.

flow Command

Use the **flow** command to display all active flows in a system.

Example: flow -l Use **-l** to list everything in the flow table. The **-l** is the only relevant debugging option.

```
# flow -l
Flow table
  Index      Source:Port      Destination:Port  Proto(V)
```

```

263484      1.1.1.252:1203      1.1.1.253:0      1 (3)
              (Action:F, S(nh):91, Statistics:22/1848)
379480      1.1.1.253:1203      1.1.1.252:0      1 (3)
              (Action:F, S(nh):75, Statistics:22/1848)

```

Each record in the flow table listing displays the index of the record, the source ip: source port, the destination ip: destination port, the inet protocol, and the source vrf to which the flow belongs.

Each new flow has to be approved by the vrouter agent. The agent does this by setting actions for each flow. There are three main actions associated with a flow table entry: Forward ('F'), Drop ('D'), and Nat ('N').

For NAT, there are additional flags indicating the type of NAT to which the flow is subject, including: SNAT (S), DNAT (D), source port translation (Ps), and destination port translation (Pd).

S(nh) indicates the source nexthop index used for the RPF check to validate that the traffic is from a known source. If the packet must go to an ECMP destination, E:X is also displayed, where 'X' indicates the destination to be used through the index within the ECMP next hop.

The Statistics field indicates the Packets/Bytes that hit this flow entry.

There is a Mirror Index field if the traffic is mirrored, listing the indices into the mirror table (which can be dumped by using **mirror --dump**).

If there is an explicit association between the forward and the reverse flows, as is the case with NAT, you will see a double arrow in each of the records with either side of the arrow displaying the flow index for that direction.

Example: flow -r Use **-r** to view all of the flow setup rates.

```

# flow -r
New = 2, Flow setup rate = 3 flows/sec, Flow rate = 3 flows/sec, for
last 548 ms
New = 2, Flow setup rate = 3 flows/sec, Flow rate = 3 flows/sec, for
last 543 ms
New = -2, Flow setup rate = -3 flows/sec, Flow rate = -3 flows/sec, for
last 541 ms
New = 2, Flow setup rate = 3 flows/sec, Flow rate = 3 flows/sec, for
last 544 ms
New = -2, Flow setup rate = -3 flows/sec, Flow rate = -3 flows/sec, for
last 542 ms

```

Example: flow --help Use **--help** to display all options available for the flow command.

```

# flow --help
Usage: flow [-f flow_index][-d flow_index][-i flow_index]
              [--mirror=mirror table index]
              [-l]
    -f <flow_index> Set forward action for flow at flow_index <flow_index>
    -d <flow_index> Set drop action for flow at flow_index <flow_index>
    -i <flow_index> Invalidate flow at flow_index <flow_index>
    --mirror          mirror index to mirror to
    -l                List all flows

```

```

-r                               Start dumping flow setup rate
--help                           Print this help

```

vrfstats Command

Use **vrfstats** to display statistics per next hop for a **vrf**. It is typically used to determine if packets are hitting the expected next hop.

Example: vrfstats --dump The **--dump** option displays the statistics for all vrfs that have seen traffic. In the following example, there was traffic only in **Vrf 0** (the public vrf). **Receives** shows the number of packets that came in the fabric destined to this location. **Encaps** shows the number of packets destined to the fabric.

If there is VM traffic going out on the fabric, the respective tunnel counters will increment.

```

# vrfstats --dump
Vrf: 0
Discards 414, Resolves 3, Receives 165334
Ecmp Composites 0, L3 Mcast Composites 0, L2 Mcast Composites 0, Fabric
Composites 0, Multi Proto Composites 0
Udp Tunnels 0, Udp Mpls Tunnels 0, Gre Mpls Tunnels 0
L2 Encaps 0, Encaps 130955

```

Example: vrfstats --get 0 Use **--get 0** to retrieve statistics for a particular **vrf**.

```

# vrfstats --get 0
Vrf: 0
Discards 418, Resolves 3, Receives 166929
Ecmp Composites 0, L3 Mcast Composites 0, L2 Mcast Composites 0, Fabric
Composites 0, Multi Proto Composites 0
Udp Tunnels 0, Udp Mpls Tunnels 0, Gre Mpls Tunnels 0
L2 Encaps 0, Encaps 132179

```

Example: vrfstats --help

```

Usage: vrfstats --get <vrf>

--dump
--help

--get <vrf>      Displays packet statistics for the vrf <vrf>
--dump           Displays packet statistics for all vrfs
--help           Displays this help message

```

rt Command

Use the **rt** command to display all routes in a **vrf**.

Example: rt --dump The following example displays **inet** family routes for **vrf 0**.

```

# rt --dump 0

Kernel IP routing table 0/0/unicast

Destination      PPL      Flags      Label      Nexthop
0.0.0.0/8        0        -          -          5

```

1.0.0.0/8	0	-	5
2.0.0.0/8	0	-	5
3.0.0.0/8	0	-	5
4.0.0.0/8	0	-	5
5.0.0.0/8	0	-	5

In this example output, the first line displays the routing table that is being dumped. In **0/0/unicast**, the first 0 is for the router id, the next 0 is for the vrf id, and unicast identifies the unicast table. The vrouter maintains separate tables for unicast and multicast routes. By default, if the **--table** option is not specified, only the unicast table is dumped.

Each record in the table output specifies the destination prefix length, the parent route prefix length from which this route has been expanded, the flags for the route, the MPLS label if the destination is a VM in another location, and the next hop id. To understand the second field "PPL", it is good to keep in mind that the unicast routing table is internally implemented as an 'mtree'.

The **Flags** field can have two values. **L** indicates that the label field is valid, and **H** indicates that **vroute** should proxy arp for this IP.

The **Nexthop** field indicates the next hop ID to which the route points.

**Example: rt --dump
--table mcst**

To dump the multicast table, use the **--table** option with **mcst** as the argument.

```
# rt --dump 0 --table mcst

Kernel IP routing table 0/0/multicast

(Src,Group)                                Nexthop

0.0.0.0,255.255.255.255
```

dropstats Command

Use the dropstats command to see packet drop counters in vrouter.

Example: dropstats

```
# dropstats

GARP                                0

ARP notme                          12904

Invalid ARPs                        0

Invalid IF                          0

Trap No IF                          0

IF TX Discard                       0

IF Drop                             49
```

IF RX Discard	0
Flow Unusable	0
Flow No Memory	0
Flow Table Full	0
Flow NAT no rflow	0
Flow Action Drop	0
Flow Action Invalid	0
Flow Invalid Protocol	0
Flow Queue Limit Exceeded	0
Discards	34
TTL Exceeded	0
Mcast Clone Fail	0
Cloned Original	0
Invalid NH	2
Invalid Label	0
Invalid Protocol	0
Rewrite Fail	0
Invalid Mcast Source	0
Push Fails	0
Pull Fails	0
Duplicated	0
Head Alloc Fails	0
Head Space Reserve Fails	0
PCOW fails	0
Invalid Packet	0
Misc	0
Nowhere to go	0
Checksum errors	0

No Fmd	0
Ivalid VNID	0
Fragment errors	0
Invalid Source	0

- dropstats ARP Block** GARP packets from VMs are dropped by vrouter, an expected behavior. In the example output, the first counter GARP indicates how many packets were dropped.
- ARP requests that are not handled by vrouter are dropped, for example, requests for a system that is not a host. These drops are counted by **ARP notme** counters.
- The **Invalid ARPs** counter is incremented when the Ethernet protocol is ARP, but the ARP operation was neither a request nor a response.
- dropstats Interface Block** **Invalid IF** counters are incremented normally during transient conditions, and should not be a concern.
- Trap No IF** counters are incremented when vrouter is not able to find the interface to trap the packets to vrouter agent, and should not happen in a working system.
- IF TX Discard** and **IF RX Discard** counters are incremented when vrouter is not in a state to transmit and receive packets, and typically happens when vrouter goes through a reset state or when the module is unloaded.
- IF Drop** counters indicate packets that are dropped in the interface layer. The increase can typically happen when interface settings are wrong.
- dropstats Flow Block** When packets go through flow processing, the first packet in a flow is cached and the vrouter agent is notified so it can take actions on the packet according to the policies configured. If more packets arrive after the first packet but before the agent makes a decision on the first packet, then those new packets are dropped. The dropped packets are tracked by the Flow unusable counter.
- The **Flow No Memory** counter increments when the flow block doesn't have enough memory to perform internal operations.
- The **Flow Table Full** counter increments when the vrouter cannot install a new flow due to lack of available slots. A particular flow can only go in certain slots, and if all those slots are occupied, packets are dropped. It is possible that the flow table is not full, but the counter might increment.
- The **Flow NAT no rflow** counter tracks packets that are dropped when there is no reverse flow associated with a forward flow that had action set as NAT. For NAT, the vrouter needs both forward and reverse flows to be set properly. If they are not set, packets are dropped.
- The **Flow Action Drop** counter tracks packets that are dropped due to policies that prohibit a flow.
- The **Flow Action Invalid** counter usually does not increment in the normal course of time, and can be ignored.
- The **Flow Invalid Protocol** usually does not increment in the normal course of time, and can be ignored.
- The **Flow Queue Limit Exceeded** usually does not increment in the normal course of time, and can be ignored.

dropstats
Miscellaneous

Operational Block The **Discard** counter tracks packets that hit a discard next hop. For various reasons interpreted by the agent and during some transient conditions, a route can point to a discard next hop. When packets hit that route, they are dropped.

The **TTL Exceeded** counter increments when the MPLS time-to-live goes to zero.

The **Mcast Clone Fail** happens when the vrouter is not able to replicate a packet for flooding.

The **Cloned Original** is an internal tracking counter. It is harmless and can be ignored.

The **Invalid NH** counter tracks the number of packets that hit a next hop that was not in a state to be used (usually in transient conditions) or a next hop that was not expected, or no next hops when there was a next hop expected. Such increments happen rarely, and should not continuously increment.

The **Invalid Label** counter tracks packets with an MPLS label unusable by vrouter because the value is not in the expected range.

The **Invalid Protocol** typically increments when the IP header is corrupt.

The **Rewrite Fail** counter tracks the number of times vrouter was not able to write next hop rewrite data to the packet.

The **Invalid Mcast Source** tracks the multicast packets that came from an unknown or unexpected source and thus were dropped.

The **Invalid Source** counter tracks the number of packets that came from an invalid or unexpected source and thus were dropped.

The remaining counters are of value only to developers.

mpls Command

The **mpls** utility command displays the input label map that has been programmed in the vrouter.

Example: mpls --dump The **--dump** command dumps the complete label map. The output is divided into two columns. The first field is the label and the second is the next hop corresponding to the label. When an MPLS packet with the specified label arrives in the vrouter, it uses the next hop corresponding to the label to forward the packet.

```
# mpls --dump
```

```
MPLS Input Label Map
```

Label	NextHop
16	9
17	11

You can inspect the operation on **nh 9** as follows:

```
# nh --get 9

Id:009  Type:Encap      Fmly: AF_INET  Flags:Valid, Policy,  Rid:0  Ref_cnt:4

      EncapFmly:0806 Oif:3 Len:14 Data:02 d0 60 aa 50 57 00 25 90 c3 08 69
      08 00
```

The nh output shows that the next hop directs the packet to go out on the interface with index 3 (**Oif:3**) with the given rewrite data.

To check the index of 3, use the following:

```
# vif -get 3

vif0/3  OS:  tapd060aa50-57

      Type:Virtual HWaddr:00:00:5e:00:01:00 IPaddr:0

      Vrf:1  Flags:PL3L2 MTU:9160 Ref:6

      RX packets:1056  bytes:103471 errors:0

      TX packets:1041  bytes:102372 errors:0
```

The **-get 3** output shows that the index of 3 corresponds to a tap interface that goes to a VM.

You can also dump individual entries in the map using the **--get** option, as follows:

```
# mpls -get 16

MPLS Input Label Map

      Label      NextHop
      -----
      16          9
```

Example: mpls -help

```
# mpls -help

Usage: mpls --dump

      mpls --get <label>

      mpls --help

--dump  Dumps the mpls incoming label map

--get    Dumps the entry corresponding to label <label>
         in the label map

--help   Prints this help message
```

mirror Command

Use the **mirror** command to dump the mirror table entries.

Example: Inspect Mirroring

The following example inspects a mirror configuration where traffic is mirrored from network **vn1 (1.1.1.0/24)** to network **vn2 (2.2.2.0/24)**. A ping is run from 1.1.1.253 to 2.2.2.253, where both IPs are valid VM IPs, then the flow table is listed:

```
# flow -l

Flow table

Index                Source:Port          Destination:Port      Proto(V)
-----
135024                2.2.2.253:1208       1.1.1.253:0          1 (1)
                        (Action:F, S(nh):17, Statistics:208/17472 Mirror Index : 0)

387324                1.1.1.253:1208       2.2.2.253:0          1 (1)
                        (Action:F, S(nh):8, Statistics:208/17472 Mirror Index : 0)
```

In the example output, **Mirror Index:0** is listed, it is the index to the mirror table. The mirror table can be dumped with the **--dump** option, as follows:

```
# mirror --dump

Mirror Table

Index  NextHop  Flags  References
-----
0      18       3
```

The mirror table entries point to next hops. In the example, the index 0 points to next hop 18. The **References** indicate the number of flow entries that point to this entry.

A next hop get operation on ID 18 is performed as follows:

```
# nh --get 18

Id:018  Type:Tunnel  Fmly: AF_INET  Flags:Valid, Udp,  Rid:0  Ref_cnt:2

  Oif:0 Len:14 Flags Valid, Udp,  Data:00 00 00 00 00 00 00 25 90 c3 08
  69 08 00

  Vrf:-1  Sip:192.168.1.10  Dip:250.250.2.253

  Sport:58818  Dport:8099
```

The **nh --get** output shows that mirrored packets go to a system with IP 250.250.2.253. The packets are tunneled as a UDP datagram and sent to the destination. **Vrf:-1** indicates that a lookup has to be done in the source **Vrf** for the destination.

You can also get an individual mirror table entry using the **--get** option, as follows:

```
# mirror --get 10

Mirror Table

Index      NextHop    Flags    References
-----
10         1          1        1
```

Example: mirror --help

```
# mirror --help

Usage: mirror --dump

        mirror --get <index>

        mirror --help

--dump  Dumps the mirror table

--get    Dumps the mirror entry corresponding to index <index>

--help   Prints this help message
```

vxlan Command

The vxlan command can be used to dump the vxlan table. The vxlan table maps a network ID to a next hop, similar to an MPLS table.

If a packet comes with a vxlan header and if the VNID is one of those in the table, the vrouter will use the next hop identified to forward the packet.

Example: vxlan --dump

```
# vxlan --dump

VXLAN Table

VNID      NextHop
-----
4         16
5         16
```

Example: vxlan --get You can use the **--get** option to dump a specific entry, as follows:

```
# vxlan --get 4

VXLAN Table

VNID      NextHop
-----
4         16
```

```

Example: vxlan --help      # vxlan --help

Usage: vxlan --dump

       vxlan --get <vnid>

       vxlan --help

--dump  Dumps the vxlan table

--get   Dumps the entry corresponding to <vnid>

--help  Prints this help message

```

nh Command

The **nh** command enables you to inspect the next hops that are known by the vrouter. Next hops tell the vrouter the next location to send a packet in the path to its final destination. The processing of the packet differs based on the type of the next hop. The next hop types are described in the following table.

Next Hop Type	Description
Receive	Indicates that the packet is destined for itself and the vrouter should perform Layer 4 protocol processing. As an example, all packets destined to the host IP will hit the receive next hop in the default VRF. Similarly, all traffic destined to the VMs hosted by the server and tunneled inside a GRE will hit the receive next hop in the default VRF first, because the outer packet that carries the traffic to the VM is that of the server.
Encap (Interface)	Used only to determine the outgoing interface and the Layer 2 information. As an example, when two VMs on the same server communicate with each other, the routes for each of them point to an encap next hop, because the only information needed is the Layer 2 information to send the packet to the tap interface of the destination VM. A packet destined to a VM hosted on one server from a VM on a different server will also hit an encap next hop, after tunnel processing.
Tunnel	Encapsulates VM traffic in a tunnel and sends it to the server that hosts the destination VM. There are different types of tunnel next hops, based on the type of tunnels used. Vrouter supports two main tunnel types for Layer 3 traffic: MPLSoGRE and MPLSoUDP. For Layer 2 traffic, a VXLAN tunnel is used. A typical tunnel next hop indicates the kind of tunnel, the rewrite information, the outgoing interface, and the source and destination server IPs.
Discard	A catch-all next hop. If there is no route for a destination, the packet hits the discard next hop, which drops the packet.
Resolve	Used by the agent to lazy install Layer 2 rewrite information.
Composite	Groups a set of next hops, called component next hops or sub next hops. Typically used when multi-destination distribution is needed, for example for multicast, ECMP, and so on.

Next Hop Type	Description
Vxlan	A VXLAN tunnel is used for Layer 2 traffic. A typical tunnel next hop indicates the kind of tunnel, the rewrite information, the outgoing interface, and the source and destination server IPs.

Example: nh --list

```

Id:000  Type:Drop      Fmly: AF_INET  Flags:Valid,  Rid:0  Ref_cnt:1781
Id:001  Type:Resolve   Fmly: AF_INET  Flags:Valid,  Rid:0  Ref_cnt:244
Id:004  Type:Receive  Fmly: AF_INET  Flags:Valid, Policy,  Rid:0
                        Ref_cnt:2 Oif:1
Id:007  Type:Encap     Fmly: AF_INET  Flags:Valid, Multicast,  Rid:0  Ref_cnt:3
                        EncapFmly:0806 Oif:3 Len:14 Data:ff ff ff ff ff ff 00 25 90 c4 82 2c
                        08 00
Id:010  Type:Encap     Fmly:AF_BRIDGE  Flags:Valid, L2,  Rid:0  Ref_cnt:3
                        EncapFmly:0000 Oif:3 Len:0 Data:
Id:012  Type:Vxlan Vrf Fmly: AF_INET  Flags:Valid,  Rid:0  Ref_cnt:2
                        Vrf:1
Id:013  Type:Composite Fmly: AF_INET  Flags:Valid, Fabric,  Rid:0  Ref_cnt:3
                        Sub NH(label): 19(1027)
Id:014  Type:Composite Fmly: AF_INET  Flags:Valid, Multicast, L3,  Rid:0
Ref_cnt:3
                        Sub NH(label): 13(0) 7(0)
Id:015  Type:Composite Fmly:AF_BRIDGE  Flags:Valid, Multicast, L2,  Rid:0
Ref_cnt:3
                        Sub NH(label): 13(0) 10(0)
Id:016  Type:Tunnel    Fmly: AF_INET  Flags:Valid, MPLSoGRE,  Rid:0  Ref_cnt:1
                        Oif:2 Len:14 Flags Valid, MPLSoGRE,  Data:00 25 90 aa 09 a6 00 25 90
                        c4 82 2c 08 00
                        Vrf:0 Sip:10.204.216.72 Dip:10.204.216.21
Id:019  Type:Tunnel    Fmly: AF_INET  Flags:Valid, MPLSoUDP,  Rid:0  Ref_cnt:7
                        Oif:2 Len:14 Flags Valid, MPLSoUDP,  Data:00 25 90 aa 09 a6 00 25 90
                        c4 82 2c 08 00
                        Vrf:0 Sip:10.204.216.72 Dip:10.204.216.21

```

```
Id:020  Type:Composite  Fmly:AF_UNSPEC  Flags:Valid, Multi Proto,  Rid:0
Ref_cnt:2
```

```
Sub NH(label): 14(0) 15(0)
```

Example: nh --get Use the **--get** option to display information for a single next hop.

```
# nh -get 9
```

```
Id:009  Type:Encap      Fmly:AF_BRIDGE  Flags:Valid, L2,  Rid:0  Ref_cnt:4
```

```
EncapFmly:0000 Oif:3 Len:0 Data:
```

Example: nh --help

```
# nh -help
```

```
Usage: nh --list
```

```
nh --get <nh_id>
```

```
nh --help
```

```
--list  Lists All Nexthops
```

```
--get   <nh_id> Displays nexthop corresponding to <nh_id>
```

```
--help  Displays this help message
```


PART 4

Monitoring and Troubleshooting Contrail

- [Configuring Traffic Mirroring to Monitor Network Traffic on page 499](#)
- [Understanding Contrail Analytics on page 515](#)
- [Configuring Contrail Analytics on page 539](#)
- [Using Contrail Analytics to Monitor and Troubleshoot the Network on page 565](#)
- [Common Support Answers on page 629](#)

CHAPTER 22

Configuring Traffic Mirroring to Monitor Network Traffic

- [Configuring Traffic Analyzers and Packet Capture for Mirroring on page 499](#)
- [Configuring Interface Monitoring and Mirroring on page 509](#)
- [Mirroring Enhancements on page 510](#)
- [Analyzer Service Virtual Machine on page 511](#)
- [Mapping VLAN Tags from a Physical NIC to a VMI \(NIC-Assisted Mirroring\) on page 513](#)

Configuring Traffic Analyzers and Packet Capture for Mirroring

Contrail provides traffic mirroring so you can mirror specified traffic to a traffic analyzer where you can perform deep traffic inspection. Traffic mirroring enables you to designate certain traffic flows to be mirrored to a traffic analyzer, where you can view traffic flows in great detail.

Use **Monitor > Debug > Packet Capture** to configure packets to be captured and “mirrored” to a virtual machine configured as a traffic analyzer. The packet activity can then be inspected for monitoring and troubleshooting purposes. This section demonstrates how to set up packet capture to mirror traffic packets to an analyzer.

- [Traffic Analyzer Images on page 499](#)
- [Configuring Traffic Analyzers on page 500](#)
- [Setting Up Traffic Mirroring Using Monitor > Debug > Packet Capture on page 500](#)
- [Setting Up Traffic Mirroring Using Configure > Networking > Services on page 504](#)

Traffic Analyzer Images

Before using the Contrail interface to configure traffic analyzers and packet capture for mirroring, make sure that the following analyzer images are available in the VM image list for your system. The traffic analyzer images are enhanced for viewing details of captured packets in Wireshark. When creating a policy for the traffic analyzer, the traffic analyzer instance should always have the **Mirror to** field selected in the policy, do not select the **Apply Service** field for a traffic analyzer.

- **analyzer-vm-console-qcow2**—Standard traffic analyzer; should be named **analyzer** in the image list. This type of traffic analyzer is always configured with a single interface, and the interface should be a **Left** interface.
- **analyzer-vm-console-two-if qcow2**—This type of traffic analyzer has two interfaces, **Left** and **Management**. This traffic analyzer can have any name except the name **analyzer**, which is reserved for the single interface analyzer.



NOTE: The analyzer-vm images are valid for all versions of Contrail. Download the images from the Contrail 1.0 software download page:
<https://www.juniper.net/support/downloads/?p=contrail#sw>.

Configuring Traffic Analyzers

In Contrail Controller, you use a two-part configuration to mirror captured packet traffic to a traffic analyzer, where the traffic details can be inspected. The configuration has the following steps:

1. Configure analyzer(s) on the host.
2. Set up rules for packet capture.

Additionally, there are two ways to configure the packet capture for the analyzers from within the Contrail interface:

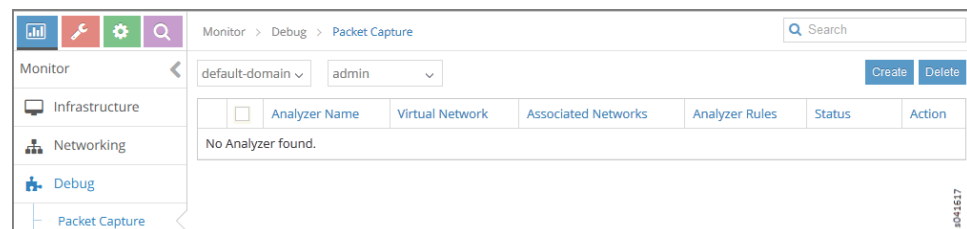
- Configure from **Monitor > Debug > Packet Capture**
- Configure from **Configure > Networking > Services**

Setting Up Traffic Mirroring Using Monitor > Debug > Packet Capture

The following are the steps needed to set up packet capture in order to “mirror” the traffic to an analyzer VM for the purpose of reviewing various aspects of packet traffic moving through the system.

1. Select **Monitor > Debug > Packet Capture**. The **Packet Capture** screen appears; see [Figure 158 on page 500](#).

Figure 158: Packet Capture



2. Click **Create** to add an analyzer; see [Figure 159 on page 501](#).

Figure 159: Create Analyzer

3. In the **Analyzer Name** field, enter a name for the analyzer and in the **Virtual Network** field, select **Automatic** or select a specific virtual network from the drop-down list of available networks; click **Save** when finished.
4. To create rules for the analyzer, in the lower portion of the **Create Analyzer** screen, click the **+** button to add a rule.

The **Analyzer Rules** fields appear; see [Figure 160 on page 501](#).

Figure 160: Analyzer Rules

5. Select the rules to apply to determine which packets should be “mirrored”—sent to the analyzer for monitoring.

See [Table 52 on page 502](#) for guidelines for completing the rule fields.

Table 52: Analyzer Rule Fields

Field	Description
IP Protocol	<p>Select from a list to define from which protocol packets are to be captured:</p> <ul style="list-style-type: none"> • ANY • TCP • UDP • ICMP
Source Network	<p>Select from a list the source network from which packets are to be captured:</p> <ul style="list-style-type: none"> • any • local • <i>domain:network 1</i> • <i>domain:network 2</i> • <i>domain:network</i>
Source Ports	<p>If you want to capture only those packets that originate from a specific port number, enter the port number.</p>
Direction	<p>Select the direction of flow for the packets to be captured:</p> <ul style="list-style-type: none"> • Bidirectional • Unidirectional
Destination Network	<p>Select from a list the destination network for the packets to be captured:</p> <ul style="list-style-type: none"> • any • local • <i>domain:network 1</i> • <i>domain:network 2</i> • <i>domain:network</i>
Destination Ports	<p>If you want to capture only those packets that are destined to a specific port number, enter the port number.</p>
Cancel, Save	<p>When finished, click Save to commit your selections, or click Cancel to clear the entries and start over.</p>

- To associate virtual networks with the analyzer, click the **Associate Networks** field in the center portion of the screen. Select from a drop-down list of available networks the networks to associate with this analyzer; see [Figure 161 on page 503](#).

Figure 161: Create Analyzer Associate Networks

Create Analyzer

Analyzer Name

demo

Virtual Network

Automatic

Associate Networks

frontend

default-virtual-network

backend

Protocol	Source Network	Source Ports	Direction	Destination Network	Destination Ports
Any	any	Source ports	<=>	any	Destination ports

Cancel

Save



NOTE: If there is already a network policy attached to the virtual network selected, any conflicting rules configured for the analyzer will not take effect.

7. View the analyzer activity from **Monitor > Debug > Packet Capture**. For the selected analyzer, click in the **Action** column and select **View Analyzer**; see [Figure 162 on page 503](#).

Figure 162: Launch Analyzer VM

Monitor > Debug > Packet Capture

default-domain

admin

Create

Delete

	Analyzer Name	Virtual Network	Associated Networks	Analyzer Rules	Status	Action
<input checked="" type="checkbox"/>	demo	Automatic	backend frontend	protocol any network default-domainadmin:frontend port any <=> network default-domainadmin:backend port any mirror_to default-domainadmin:demo	Active	<div><div>Edit</div><div>View Analyzer</div><div>Delete</div></div>

Associated Networks:

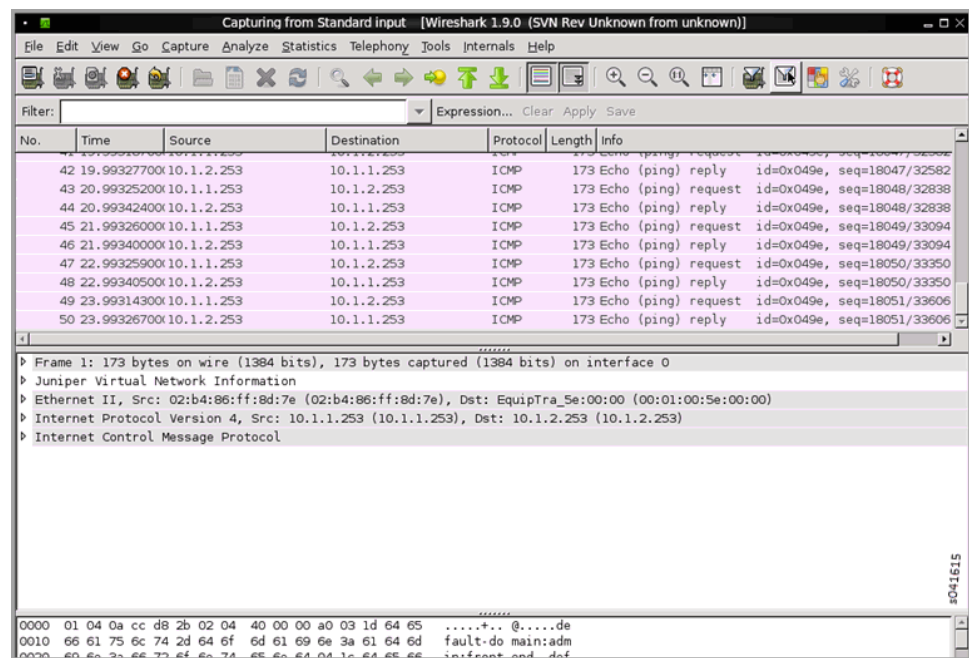
backend frontend

Analyzer Rules:

protocol any network default-domainadmin:frontend port any <=> network default-domainadmin:backend port any mirror_to default-domainadmin:demo

8. The Wireshark **Packet Capture Display** appears; see [Figure 163 on page 504](#).

Figure 163: Packet Capture Display



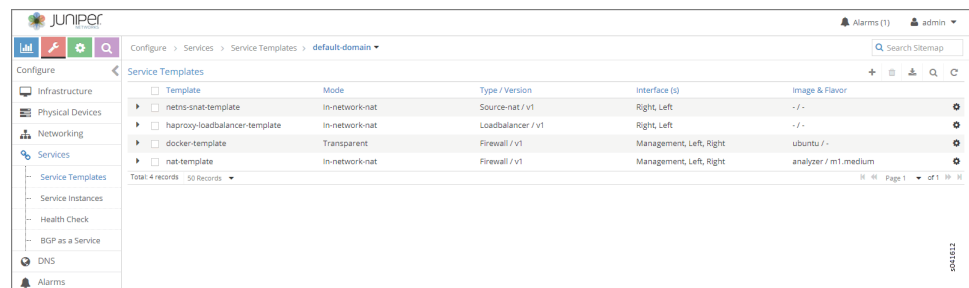
Setting Up Traffic Mirroring Using Configure > Networking > Services

You can set up packet capture for mirroring to an analyzer within a service chain utilizing more than one interface by starting with a service template. The following procedure provides the steps needed.

1. Access **Configure > Services > Service Templates**.

The **Service Templates** screen appears; see [Figure 164 on page 504](#).

Figure 164: Service Templates



2. To create a new service template, click the **+** icon.

The **Create** window appears. Select the **Service Template** tab; see [Figure 165 on page 505](#).

Figure 165: Create Service Template

Create

Service Template

Permissions

Name

analyzer-service-template

Version

v2

Virtualization Type

Virtual Machine

Service Mode

Transparent

Service Type

Analyzer

Interface (s)

management

left

Cancel

Save

3. Complete the fields by using the guidelines in [Table 53 on page 505](#).

Table 53: Create Service Template Fields

Field	Description
Name	Enter a descriptive text name for this service template.
Version	Select v2 from the drop-down list to indicate that this service template is based on templates version 2, valid for Contrail 3.0 and later.
Virtualization Type	Select Virtual Machine from the drop-down list to indicate the virtualization type for mirroring for this template.
Service Mode	Select Transparent from the drop-down list to indicate that this service template is for transparent mirroring.
Service Type	Select Analyzer from the drop-down list to indicate that this service template is for a traffic analyzer.
Interface(s)	From the drop-down list, click the check boxes to indicate which interface types are used for this analyzer service template: <ul style="list-style-type: none">• Left• Right• Management
Save	When finished, click OK to commit the changes

Table 53: Create Service Template Fields (*continued*)

Field	Description
Cancel	Click Cancel to clear the fields and start over.

4. Create a service instance by clicking the **Service Instances** link and clicking the **+** icon.
The **Create** window appears; make sure the Service Instance tab is selected. See [Figure 166 on page 506](#).

Figure 166: Create Service Instances

5. Complete the fields by using the guidelines in [Table 54 on page 506](#).

Table 54: Create Service Instances Fields

Field	Description
Name	Enter a text name for this service instance.
Service Template	Select from a drop-down list of available service templates the template to use for this service instance, analyzer-service-template in this example.
Interface Type	Each interface configured in the service template for this instance appears in a list.
Virtual Network	Select from a drop-down list of available virtual networks the network for each interface that is configured for the instance.

Table 54: Create Service Instances Fields (continued)

Field	Description
Save	Click Save to commit your changes.
Cancel	Click Cancel to clear your changes and start over.

6. To create a network policy rule for this service instance, click **Configure > Networking > Policies**. The **Policies** window appears. Click the + icon to get to the **Create** window; see [Figure 167 on page 507](#).

Figure 167: Create Policy

- 7.
8. Enter a name for the policy, then click the + icon in the lower portion of the screen to configure rules for the policy, see [Figure 168 on page 507](#).

Figure 168: Create Policy Rules

9. To add policy rules, complete the fields, using the guidelines in [Table 55 on page 508](#).



NOTE: When there is a network policy attached to the virtual network, any conflicting rules configured for the analyzer will not take effect.

Table 55: Add Rule Fields

Field	Description
Action	Select PASS or DENY as the rule action.
Protocol	Select the protocol for the policy rule, or select ANY.
Source	Select from multiple drop-down lists the source for this rule, including options under CIDR, Network, Policy, or Security Group.
Ports	Select from a drop-down list the source ports for the rule.
Direction	Select the direction of flow for the packets to be captured: <ul style="list-style-type: none"> <> (bidirectional) > (unidirectional)
Destination	Select from multiple drop-down lists the destination for this rule, including options under CIDR, Network, Policy, or Security Group.
Ports	Select from a list the destination ports for the packets to be captured.
check boxes	Check any box that applies to this rule: Log, Services, Mirror, QoS.
Save	Click Save to commit your changes.
Cancel	Click Cancel to clear your changes and start over.

10. When finished, click **Save**.

11. To verify packet capture, at **Configure > Services > Service Instances**, select the analyzer service instance and click **View Console**.

The packet capture displays; see [Figure 169 on page 508](#). The analyzer service VM launches the Contrail-enhanced Wireshark as it starts and captures the mirrored packets destined to this service.

Figure 169: Service Instances View Console



- See Also**
- [Configuring Interface Monitoring and Mirroring on page 509](#)
 - [Mirroring Enhancements on page 510](#)

- [Analyzer Service Virtual Machine on page 511](#)
- [Mapping VLAN Tags from a Physical NIC to a VMI \(NIC-Assisted Mirroring\) on page 513](#)

Configuring Interface Monitoring and Mirroring

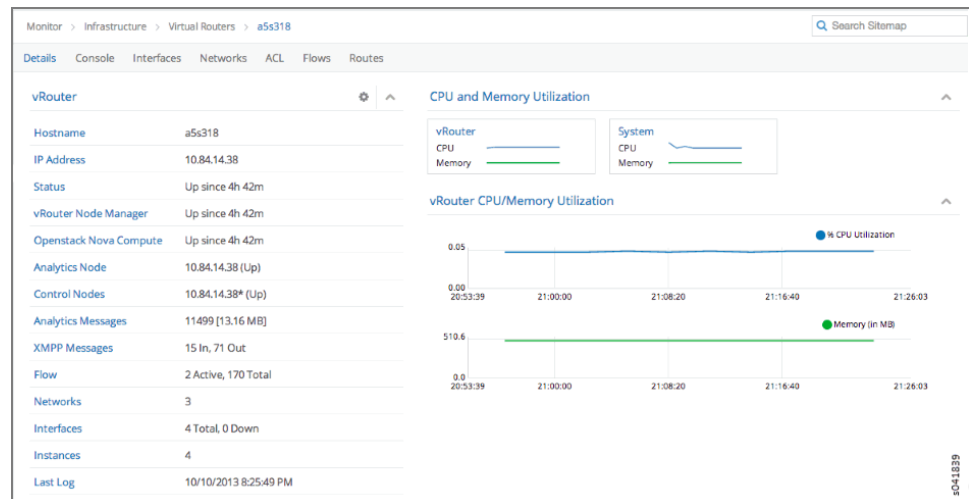
Contrail supports user monitoring of traffic on any guest virtual machine interface when using the Juniper Contrail user interface.

When interface monitoring (packet capture) is selected, a default analyzer is created and all traffic from the selected interface is mirrored and sent to the default analyzer. If a mirroring instance is already launched, the traffic will be redirected to the selected instance. The interface traffic is only mirrored during the time that the monitor packet capture interface is in use. When the capture screen is closed, interface mirroring stops.

To configure interface mirroring:

1. Select **Monitor > Infrastructure > Virtual Routers**, then select the vRouter that has the interface to mirror.
2. In the list of attributes for the vRouter, select **Interfaces**; see [Figure 170 on page 509](#).

Figure 170: Individual vRouter



A list of interfaces for that vRouter appears.

3. For the interface to mirror, click the Action icon in the last column and select the option **Packet Capture**; see [Figure 171 on page 510](#).

Figure 171: Interfaces

Name	Label	Status	Network	IP Address	Floating IP	Instance
tap32e8b4a6-4b	19	Up	default-domain:demo:svc-vn-left	250.250.2.252	None	2797154f-aa4e-4e1a-a4dc-e7aa95a5a4f3
tap347d03ac-c2	16	Up	default-domain:demo:svc-vn-left	250.250.2.253	None	27b87f5-5dc2-4b49-bd08-b15e96c311b3
tap53aef323-79	17	Up	default-domain:demo:fe	7.7.7.253	None	cee63d6b-8843-4382-a699-fd01e85781b9
tap96b4d43c-75	18	Up	default-domain:demo:be	8.8.8.253	None	26c28bb7-0259-42ee-ad92-0c29c182dbf6

The mirror packet capture starts and displays at this screen.

The mirror packet capture stops when you exit this screen.

Related Documentation

- [Configuring Traffic Analyzers and Packet Capture for Mirroring on page 499](#)
- [Mirroring Enhancements on page 510](#)
- [Analyzer Service Virtual Machine on page 511](#)
- [Mapping VLAN Tags from a Physical NIC to a VMI \(NIC-Assisted Mirroring\) on page 513](#)

Mirroring Enhancements

Mirroring Specified Traffic

Specific traffic can be mirrored to a traffic analyzer in Contrail by:

- Configuring rules to identify the flows to be mirrored, and
- Specifying the analyzer to which the traffic is mirrored

Additionally, mirroring can be configured on virtual machine (VM) interfaces to send all the traffic to and from the interface to the specified analyzer.

Configuring Headers and Next Hops

When a packet is mirrored, a Juniper header is added to provide additional information in the analyzer, then the packet is encapsulated and sent to the destination.

Starting with Contrail 3.x releases, mirroring is enhanced with the following options:

- Option to control addition of the Juniper header in the mirrored packet.
 - When disabled, the Juniper header is not added to the mirrored packet.
- Option to control whether the next hop used is dynamic or static.
 - If dynamic is selected, the next hop based on the destination is used. Packets are forwarded to the destination based on the encapsulation priority.
 - If static is chosen, the next hop is created for the specified destination with VxLAN encapsulation using the configured VNI, destination VTEP, and MAC to transmit the mirrored packets.

The following combinations are supported:

- Dynamic next hop with Juniper header added
The default combination and the only supported case up to Release 3.0.2
- Dynamic next hop, without Juniper header
- Static next hop, without Juniper header, with the original Layer 2 packet

How Mirroring is Implemented

The Contrail vrouter agent adds a mirror entry in the vrouter and points to the next hop to be used. The data for the Juniper header is taken from the flow entry. For interface mirroring, the Juniper header has a TLV in the metadata to use the interface name instead of providing a destination VN.

For more information about implementation details, see <https://github.com/Juniper/contrail-controller/wiki/Mirroring>.

Related Documentation

- [Configuring Traffic Analyzers and Packet Capture for Mirroring on page 499](#)
- [Configuring Interface Monitoring and Mirroring on page 509](#)
- [Analyzer Service Virtual Machine on page 511](#)
- [Mapping VLAN Tags from a Physical NIC to a VMI \(NIC-Assisted Mirroring\) on page 513](#)

Analyzer Service Virtual Machine

The analyzer service virtual machine (**analyzer-vm-console.qcow2**) launches a Contrail-enhanced version of the network protocol analyzer Wireshark as the analyzer starts capturing mirror packets destined to the analyzer service.

- [Packet Format for Analyzer on page 511](#)
- [Metadata Format on page 512](#)
- [Wireshark Changes on page 513](#)
- [Troubleshooting Packet Display on page 513](#)

Packet Format for Analyzer

The analyzer uses the PCAP format, which has these parts:

- Global header
- PCAP packet header
- Packet data (original packet data)

The global header is added by the analyzer service by means of the Wireshark instance. The vRouter DP uses the configured UDP session to send mirrored packets to the analyzer, adding the PCAP packet header to the packet data as it sends it over the UDP socket to the analyzer.

The following additional information is also added to the packet data as metadata:

- Captured host (IP address)
- Ingress or egress
- Action (Pass/Deny/...)
- Source VN (fully qualified name)
- Destination VN (fully qualified name)

In the existing PCAP, a network ID is added in the global header. The metadata (additional flow information) is added in front of the existing packet as follows.

```
+++++
| Global header | Packet header| Meta data |Packet data| Packet header| Meta data
|Packet data|
+++++
```

Metadata Format

The metadata is in type-length-value (TLV) format as follows.

Type: 1 Byte

Length: 1 Byte

Value: up to length

Type

1 – Captured host IPv4 address

2 - Action field

3 – Source VN

4 – Destination VN

255 – TLV end

Captured host address

Length is 4 or 16 bytes based on IP address type

Action field

Length is 2 bytes. Multiple bits might be turned on, if there are more actions. Ingress or egress bit will be present in the Action field.

Source VN or Destination VN

Length is variable and up to 256 characters

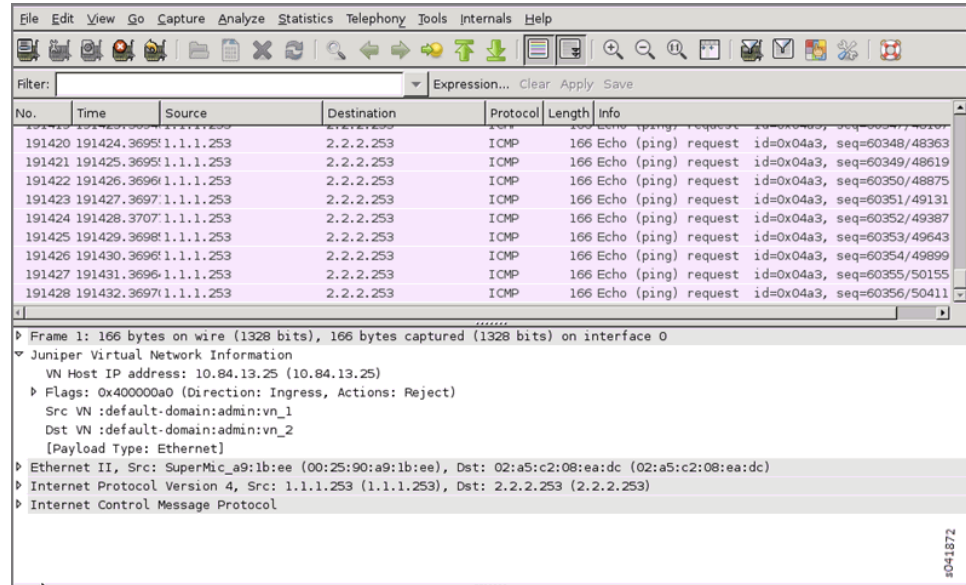
TLV end

A special type **255 (0xFF)** is used to identify the end of TLV entries. The TLV end must be last, at the end of the metadata.

Wireshark Changes

A plugin is added to the Wireshark code. The plugin parses the metadata and displays the packet fields; see example in [Figure 172 on page 513](#).

Figure 172: Wireshark Packet Display



Troubleshooting Packet Display

Follow these steps if the packets are not displaying:

1. Use **tcpdump** on the tap interfaces to see if packets are going towards the analyzer VM.
2. Check introspect to see whether the flow action has mirror activity in it or not.

Related Documentation

- [Configuring Traffic Analyzers and Packet Capture for Mirroring on page 499](#)
- [Configuring Interface Monitoring and Mirroring on page 509](#)
- [Mirroring Enhancements on page 510](#)
- [Mapping VLAN Tags from a Physical NIC to a VMI \(NIC-Assisted Mirroring\) on page 513](#)

Mapping VLAN Tags from a Physical NIC to a VMI (NIC-Assisted Mirroring)

When mirroring is enabled, the vRouter throughput reduces because of the additional packet handling overhead caused by cloning the packet to be mirrored, encapsulating it

in the required header, and forwarding it to the mirror destination. Impact to throughput increases in proportion to the amount of traffic that needs to be mirrored.

A solution to avoid impact on throughput due to mirroring is to use the mirroring capabilities of an installed Network Interface Card (NIC).

Contrail Release 4.0 has the ability to mirror specific traffic to a traffic analyzer or to a physical probe using the Network interface card (NIC) instead of the vRouter to mirror packets. When NIC-assisted mirroring is enabled, ingress packets to be mirrored sent from a VM are routed to the NIC with a configured VLAN tag. The NIC is configured for VLAN port-mirroring and mirrors any packet with the VLAN tag.

In this approach, the vRouter doesn't mirror the packets. When NIC-assisted mirroring is enabled, the ingress packets coming from the VM that are to be mirrored are sent to the NIC with a configured VLAN tag.

The NIC is programmed to do VLAN port mirroring, so that any packet with the configured VLAN is mirrored additionally by the NIC. This change in vRouter is only for traffic coming from the VMs. Traffic coming from the fabric is directly mirrored from the NIC itself and there is no additional mirroring need in vRouter. The programming of the NIC itself for appropriate mirroring is outside the scope of the current activity. An example is the Niantic 82599 10G NIC, which supports VLAN port mirroring options.

The following are cautions to observe when using NIC-assisted mirroring:

- VM traffic sent to another VM running on the same compute node will not be mirrored when NIC-assisted mirroring is selected.
- Traffic coming in from the fabric interface will not be mirrored.
- When a VLAN interface is used as the fabric interface, traffic will be tagged first with the NIC-assisted mirroring VLAN, followed by the VLAN tag on the fabric interface. The NIC-assisted mirroring VLAN will be the inner tag and the fabric interface VLAN will be the outer tag.

The NIC must be programmed for VLAN port mirroring. While configuring mirroring in Contrail, the user can indicate NIC-assisted mirroring with the VLAN tag. The Contrail UI supports NIC-assisted mirroring configuration in the Ports page and in the Policies page with an additional flag for NIC-assisted mirroring and the VLAN tag to be used.

**Related
Documentation**

- [Configuring Traffic Analyzers and Packet Capture for Mirroring on page 499](#)
- [Configuring Interface Monitoring and Mirroring on page 509](#)
- [Mirroring Enhancements on page 510](#)
- [Analyzer Service Virtual Machine on page 511](#)

Understanding Contrail Analytics

- [Understanding Contrail Analytics on page 515](#)
- [Contrail Alerts on page 516](#)
- [Underlay Overlay Mapping in Contrail on page 519](#)

Understanding Contrail Analytics

Contrail is a distributed system of compute nodes, control nodes, configuration nodes, database nodes, web UI nodes, and analytics nodes.

The analytics nodes are responsible for the collection of system state information, usage statistics, and debug information from all of the software modules across all of the nodes of the system. The analytics nodes store the data gathered across the system in a database that is based on the Apache Cassandra open source distributed database management system. The database is queried by means of an SQL-like language and representational state transfer (REST) APIs.

System state information collected by the analytics nodes is aggregated across all of the nodes, and comprehensive graphical views allow the user to get up-to-date system usage information easily.

Debug information collected by the analytics nodes includes the following types:

- System log (syslog) messages—informational and debug messages generated by system software components.
- Object log messages—records of changes made to system objects such as virtual machines, virtual networks, service instances, virtual routers, BGP peers, routing instances, and the like.
- Trace messages—records of activities collected locally by software components and sent to analytics nodes only on demand.

Statistics information related to flows, CPU and memory usage, and the like is also collected by the analytics nodes and can be queried at the user interface to provide historical analytics and time-series information. The queries are performed using REST APIs.

Analytics data is written to a database in Contrail. The data expires after the default time-to-live (TTL) period of 48 hours. This default TTL time can be changed as needed by changing the value of the `database_ttl` value in the cluster configuration.

**Related
Documentation**

- [Contrail Alerts on page 516](#)
- [Analytics Scalability on page 540](#)
- [High Availability for Analytics on page 541](#)
- [Ceilometer Support in a Contrail Cloud on page 544](#)
- [Underlay Overlay Mapping in Contrail on page 519](#)
- [Monitoring the System on page 566](#)
- [Debugging Processes Using the Contrail Introspect Feature on page 568](#)
- [Monitor > Infrastructure > Dashboard on page 572](#)
- [Monitor > Infrastructure > Control Nodes on page 574](#)
- [Monitor > Infrastructure > Virtual Routers on page 581](#)
- [Monitor > Infrastructure > Analytics Nodes on page 591](#)
- [Monitor > Infrastructure > Config Nodes on page 596](#)
- [Monitor > Networking on page 599](#)
- [Understanding Flow Sampling on page 619](#)
- [Query > Flows on page 607](#)
- [Query > Logs on page 614](#)
- [System Log Receiver in Contrail Analytics on page 542](#)
- [Example: Debugging Connectivity Using Monitoring for Troubleshooting on page 622](#)

Contrail Alerts

Starting with Contrail 3.0 and greater, Contrail alerts are provided on a per-user visible entity (UVE) basis.

Contrail analytics raise or clear alerts using Python-coded rules that examine the contents of the UVE and the configuration of the object. Some rules are built in. Others can be added using Python *stevedore* plugins.

This topic describes Contrail alerts capabilities.

Alert API Format

The Contrail alert analytics API provides the following:

- Read access to the alerts as part of the UVE GET APIs.
- Alert acknowledgement using POST requests.

- UVE and alert streaming using server-sent events (SSEs).

For example:

GET `http://<analytics-ip>:8081/analytics/uves/control-node/a6s40?flat`

```
{
  NodeStatus: {...},
  ControlCpuState: {...},
  UVEAlarms: {
    alarms: [
      {
        description: [
          {
            value: "0 != 2",
            rule: "BgpRouterState.num_up_bgp_peer !=
BgpRouterState.num_bgp_peer"
          }
        ],
        ack: false,
        timestamp: 1442995349253178,
        token: "eyJ0aW1lc3RhbnRhaXN0YyOTk1MzQ5MTUzMTc4LCAiaHR0cF9wb3J0Ijog
NTk5NSwgImhvc3RfaXAiOiAiMTAuODQuMTMuNDUifQ==",
        type: "BgpConnectivity",
        severity: 4
      }
    ]
  },
  BgpRouterState: {...}
}
```

In the example:

- Alerts are raised on a per-UVE basis and can be retrieved by a GET on a UVE.
- An **ack** indicates if the alert has been acknowledged or not.
- A **token** is used by clients when requesting acknowledgements

Analytics APIs for Alerts

The following examples show the API to use to display alerts and alarms and to acknowledge alarms.

- To retrieve a list of alerts raised against the control node named **aXXsYY**.

```
GET
http://<analytics-ip>:<rest-api-port>/analytics/uves/control-node/aXXsYY&filter=UVEAlarms
```

This is available for all UVE table types.

- To retrieve a list of all alarms in the system.

```
GET http://<analytics-ip>:<rest-api-port>/analytics/alarms
```

- To acknowledge an alarm.

```
POST http://<analytics-ip>:<rest-api-port>/analytics/alarms/acknowledge
Body: {"table": <object-type>, "name": <key>, "type": <alarm type>, "token":
<token>}
```

Acknowledged and unacknowledged alarms can be queried specifically using the following URL query parameters along with the GET operations listed previously.

```
ackFilt=True
ackFilt=False
```

Analytics APIs for SSE Streaming

The following examples show the API to use to retrieve all or portions of SE streams.

- To retrieve an SSE-based stream of UVE updates for the control node alarms.

```
GET
http://<analytics-ip>:<rest-api-port>/analytics/uve-stream?tablefilt=control-node
```

This is available for all UVE table types. If the **tablefilt** URL query parameter is not provided, all UVEs are retrieved.

- To retrieve only the alerts portion of the SSE-based stream of UVE updates instead of the entire content.

```
GET
http://<analytics-ip>:<rest-api-port>/analytics/alarms-stream?tablefilt=control-node
```

This is available for all UVE table types. If the **tablefilt** URL query parameter is not provided, all UVEs are retrieved.

Built-in Node Alerts

The following built-in node alerts can be retrieved using the APIs listed in *Analytics APIs for Alerts*.

```
control node: {
  PartialSysinfoControl: "Basic System Information is absent for this node in
  BgpRouterState.build_info",
  ProcessStatus: "NodeMgr reports abnormal status for process(es) in
  NodeStatus.process_info",
  XmppConnectivity: "Not enough XMPP peers are up in
  BgpRouterState.num_up_bgp_peer",
  BgpConnectivity: "Not enough BGP peers are up in
  BgpRouterState.num_up_bgp_peer",
  AddressMismatch: "Mismatch between configured IP Address and operational IP
  Address",
  ProcessConnectivity: "Process(es) are reporting non functional components in
  NodeStatus.process_status"
},

vrouter: {
  PartialSysinfoCompute: "Basic System Information is absent for this node in
  VrouterAgent.build_info",
  ProcessStatus: "NodeMgr reports abnormal status for process(es) in
  NodeStatus.process_info",
  ProcessConnectivity: "Process(es) are reporting non functional components in
  NodeStatus.process_status",
  VrouterInterface: "VrouterAgent has interfaces in error state in
  VrouterAgent.error_intf_list",
  VrouterConfigAbsent: "Vrouter is not present in Configuration",
},

config node: {
```

```
PartialSysinfoConfig: "Basic System Information is absent for this node in
ModuleCpuState.build_info",
ProcessStatus: "NodeMgr reports abnormal status for process(es) in
NodeStatus.process_info",
ProcessConnectivity: "Process(es) are reporting non functional components in
NodeStatus.process_status"
},

analytics node: {
ProcessStatus: "NodeMgr reports abnormal status for process(es) in
NodeStatus.process_info"
PartialSysinfoAnalytics: "Basic System Information is absent for this node in
CollectorState.build_info",
ProcessConnectivity: "Process(es) are reporting non functional components in
NodeStatus.process_status"
},

database node: {
ProcessStatus: "NodeMgr reports abnormal status for process(es) in
NodeStatus.process_info",
ProcessConnectivity: "Process(es) are reporting non functional components in
NodeStatus.process_status"
},
```

**Related
Documentation**

- [Monitoring the System on page 566](#)
- [Debugging Processes Using the Contrail Introspect Feature on page 568](#)
- [Monitor > Infrastructure > Dashboard on page 572](#)
- [Monitor > Infrastructure > Control Nodes on page 574](#)
- [Monitor > Infrastructure > Virtual Routers on page 581](#)
- [Monitor > Infrastructure > Analytics Nodes on page 591](#)
- [Monitor > Infrastructure > Config Nodes on page 596](#)
- [Monitor > Networking on page 599](#)
- [Understanding Flow Sampling on page 619](#)
- [Query > Flows on page 607](#)
- [Query > Logs on page 614](#)
- [Example: Debugging Connectivity Using Monitoring for Troubleshooting on page 622](#)

Underlay Overlay Mapping in Contrail

- [Overview: Underlay Overlay Mapping using Contrail Analytics on page 520](#)
- [Underlay Overlay Analytics Available in Contrail on page 520](#)
- [Architecture and Data Collection on page 521](#)
- [New Processes/Services for Underlay Overlay Mapping on page 521](#)
- [External Interfaces Configuration for Underlay Overlay Mapping on page 522](#)
- [Physical Topology on page 522](#)

- [SNMP Configuration on page 523](#)
- [Link Layer Discovery Protocol \(LLDP\) Configuration on page 523](#)
- [IPFIX and sFlow Configuration on page 523](#)
- [Sending pRouter Information to the SNMP Collector in Contrail on page 525](#)
- [pRouter UVEs on page 525](#)
- [Contrail User Interface for Underlay Overlay Analytics on page 527](#)
- [Enabling Physical Topology on the Web UI on page 527](#)
- [Viewing Topology to the Virtual Machine Level on page 528](#)
- [Viewing the Traffic of any Link on page 528](#)
- [Trace Flows on page 529](#)
- [Search Flows and Map Flows on page 530](#)
- [Overlay to Underlay Flow Map Schemas on page 530](#)
- [Module Operations for Overlay Underlay Mapping on page 533](#)
- [SNMP Collector Operation on page 533](#)
- [Topology Module Operation on page 534](#)
- [IPFIX and sFlow Collector Operation on page 535](#)
- [Troubleshooting Underlay Overlay Mapping on page 535](#)
- [Script to add pRouter Objects on page 536](#)

Overview: Underlay Overlay Mapping using Contrail Analytics

Today's cloud data centers consist of large collections of interconnected servers that provide computing and storage capacity to run a variety of applications. The servers are connected with redundant TOR switches, which in turn, are connected to spine routers. The cloud deployment is typically shared by multiple tenants, each of whom usually needs multiple isolated networks. Multiple isolated networks can be provided by overlay networks that are created by forming tunnels (for example, gre, ip-in-ip, mac-in-mac) over the underlay or physical connectivity.

As data flows in the overlay network, Contrail can provide statistics and visualization of the traffic in the underlay network.

Underlay Overlay Analytics Available in Contrail

Starting with Contrail Release 2.20, you can view a variety of analytics related to underlay and overlay traffic in the Contrail Web user interface. The following are some of the analytics that Contrail provides for statistics and visualization of overlay underlay traffic.

- View the topology of the underlay network.

A user interface view of the physical underlay network with a drill down mechanism to show connected servers (contrail computes) and virtual machines on the servers.
- View the details of any element in the topology.

You can view details of a pRouter, vRouter, or virtual machine link between two elements. You can also view traffic statistics in a graphical view corresponding to the selected element.

- View the underlay path of an overlay flow.

Given an overlay flow, you can get the underlay path used for that flow and map the path in the topology view.

Architecture and Data Collection

Accumulation of the data to map an overlay flow to its underlay path is performed in several steps across Contrail modules.

The following outlines the essential steps:

1. The SNMP collector module polls physical routers.

The SNMP collector module receives the authorizations and configurations of the physical routers from the Contrail config module, and polls all of the physical routers, using SNMP protocol. The collector uploads the data to the Contrail analytics collectors. The SNMP information is stored in the pRouter UVEs (physical router user visible entities).

2. IPFIX and sFlow protocols are used to collect the flow statistics.

The physical router is configured to send flow statistics to the collector, using one of the collection protocols: Internet Protocol Flow Information Export (IPFIX) or sFlow (an industry standard for sampled flow of packet export at Layer 2).

3. The topology module reads the SNMP information.

The Contrail topology module reads SNMP information from the pRouter UVEs from the analytics API, computes the neighbor list, and writes the neighbor information into the pRouter UVEs. This neighbor list is used by the Contrail WebUI to display the physical topology.

4. The Contrail user interface reads and displays the topology and statistics.

The Contrail user interface module reads the topology information from the Contrail analytics and displays the physical topology. It also uses information stored in the analytics to display graphs for link statistics, and to show the map of the overlay flows on the underlay network.

New Processes/Services for Underlay Overlay Mapping

The **contrail-snmp-collector** and the **contrail-topology** are new daemons that are both added to the **contrail-analytics** node. The **contrail-analytics** package contains these new features and their associated files. The **contrail-status** displays the new services.

Example: The following is an example of using **contrail-status** to show the status of the new process and service for underlay overlay mapping.

```
user@host:~# contrail-status

== Contrail Control ==

supervisor-control:      active
contrail-control         active

...

== Contrail Analytics ==

supervisor-analytics:    active

...

contrail-query-engine     active
contrail-snmp-collector   active
contrail-topology         active
```

Example: Service Command The **service** command can be used to start, stop, and restart the new services. See the following example.

```
user@host:~# service contrail-snmp-collector status

contrail-snmp-collector    RUNNING   pid 12179, uptime 1 day, 14:59:11
```

External Interfaces Configuration for Underlay Overlay Mapping

This section outlines the external interface configurations necessary for successful underlay overlay mapping for Contrail analytics.

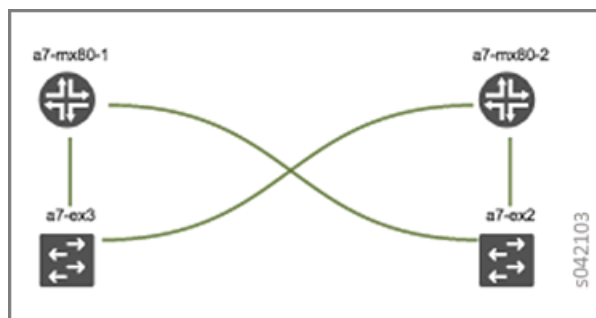
Physical Topology

The typical physical topology includes:

- Servers connected to the ToR switches.
- ToR switches connected to spine switches.
- Spine switches connected to core switches.

The following is an example of how the topology is depicted in the Contrail WebUI analytics.

Figure 173: Analytics Topology



SNMP Configuration

Configure SNMP on the physical devices so that the **contrail-snmp-collector** can read SNMP data.

The following shows an example SNMP configuration from a Juniper Networks device.

```
set snmp community public authorization read-only
```

Link Layer Discovery Protocol (LLDP) Configuration

Configure LLDP on the physical device so that the **contrail-snmp-collector** can read the neighbor information of the routers.

The following is an example of LLDP configuration on a Juniper Networks device.

```
set protocols lldp interface all
```

```
set protocols lldp-med interface all
```

IPFIX and sFlow Configuration

Flow samples are sent to the **contrail-collector** by the physical devices. Because the **contrail-collector** supports the sFlow and IPFIX protocols for receiving flow samples, the physical devices, such as MX Series devices or ToR switches, must be configured to send samples using one of those protocols.

Example: sFlow Configuration

The following shows a sample sFlow configuration. In the sample, the IP variable *<source ip>* refers to the loopback or IP that can be reachable of the device that acts as an sflow source, and the other IP variable *<collector_IP_data>* is the address of the collector device.

```

root@host> show configuration protocols sflow | display set

set protocols sflow polling-interval 0

set protocols sflow sample-rate ingress 10

set protocols sflow source-ip <source ip>4

set protocols sflow collector <collector_IP_data> udp-port 6343

set protocols sflow interfaces ge-0/0/0.0

```

```
set protocols sflow interfaces ge-0/0/1.0
set protocols sflow interfaces ge-0/0/2.0
set protocols sflow interfaces ge-0/0/3.0
set protocols sflow interfaces ge-0/0/4.0
```

Example: IPFIX Configuration

The following is a sample IPFIX configuration from a Juniper Networks device. The IP address variable `<ip_sflow collector>` represents the sflow collector (control-collector analytics node) and `<source ip>` represents the source (outgoing) interface on the router/switch device used for sending flow data to the collector. This could also be the lo0 address, if it is reachable from the Contrail cluster.

```
root@host> show configuration chassis | display set

set chassis tfeb slot 0 sampling-instance sample-ins1

set chassis network-services

root@host> show configuration chassis tfeb | display set

set chassis tfeb slot 0 sampling-instance sample-ins1

root@host > show configuration services flow-monitoring | display set

set services flow-monitoring version-ipfix template t1 flow-active-timeout 30
set services flow-monitoring version-ipfix template t1 flow-inactive-timeout 30
set services flow-monitoring version-ipfix template t1 template-refresh-rate packets 10
set services flow-monitoring version-ipfix template t1 ipv4-template

root@host > show configuration interfaces | display set | match sampling

set interfaces ge-1/0/0 unit 0 family inet sampling input
set interfaces ge-1/0/1 unit 0 family inet sampling input

root@host> show configuration forwarding-options sampling | display set

set forwarding-options sampling instance sample-ins1 input rate 1

set forwarding-options sampling instance sample-ins1 family inet output
flow-server <ip_sflow collector> port 4739

set forwarding-options sampling instance sample-ins1 family inet output
flow-server <ip_sflow collector> version-ipfix template t1
```

```
set forwarding-options sampling instance sample-ins1 family inet output
inline-jflow source-address <source ip>
```

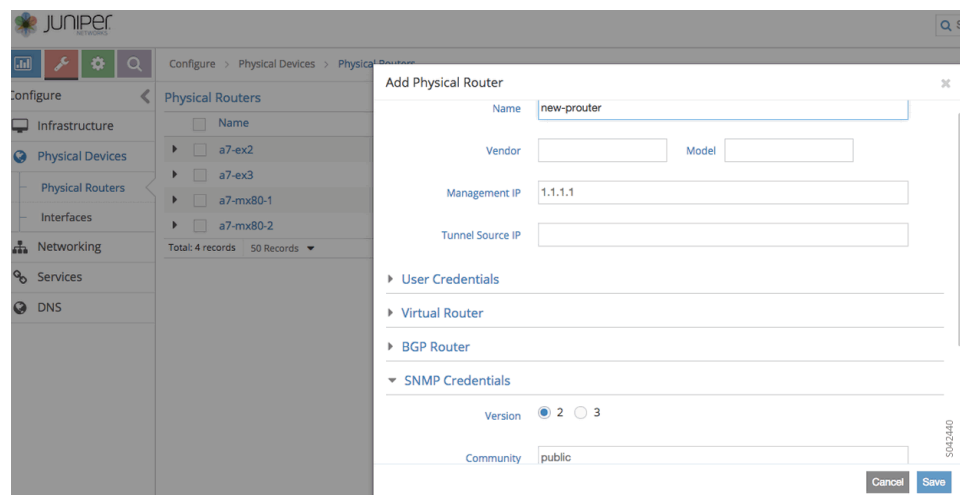
Sending pRouter Information to the SNMP Collector in Contrail

Information about the physical devices must be sent to the SNMP collector before the full analytics information can be read and displayed. Typically, the pRouter information is taken from the **contrail-config** file.

SNMP collector getting pRouter information from contrail-config file

The physical routers are added to the **contrail-config** by using the Contrail user interface or by using direct API, by means of provisioning or other scripts. Once the configuration is in the **contrail-config**, the **contrail-snmp-collector** gets the physical router information from **contrail-config**. The SNMP collector uses this list and the other configuration parameters to perform SNMP queries and to populate pRouter UVEs.

Figure 174: Add Physical Router Window



pRouter UVEs

pRouter UVEs are accessed from the REST APIs on your system from **contrail-analytics-api**, using a URL of the form:

http://<host ip>:8081/analytics/uves/prouters

The following is sample output from a pRouter REST API:

Figure 175: Sample Output From a pRouter REST API

```
[
  - {
    href: "http://10.84.63.130:8081/analytics/uves/prouter/a7-mx80-1?flat",
    name: "a7-mx80-1"
  },
  - {
    href: "http://10.84.63.130:8081/analytics/uves/prouter/a7-mx80-2?flat",
    name: "a7-mx80-2"
  },
  - {
    href: "http://10.84.63.130:8081/analytics/uves/prouter/a7-ex3?flat",
    name: "a7-ex3"
  },
  - {
    href: "http://10.84.63.130:8081/analytics/uves/prouter/a7-ex2?flat",
    name: "a7-ex2"
  }
]
```

s042104

Details of a pRouter UVE can be obtained from your system, using a URL of the following form:

http://<host ip>:8081/analytics/uves/prouter/a7-ex3?flat

The following is sample output of a pRouter UVE.

Figure 176: Sample Output From a pRouter UVE

```

{
  - PRouterFlowEntry: {
    flow_export_source_ip: "10.84.63.114"
  },
  - PRouterLinkEntry: {
    - link_table: [
      - {
        remote_interface_name: "ge-1/0/1",
        local_interface_name: "ge-0/0/0.0",
        remote_interface_index: 517,
        local_interface_index: 503,
        type: 1,
        remote_system_name: "a7-mx80-1"
      },
      - {
        remote_interface_name: "ge-1/0/1",
        local_interface_name: "ge-0/0/1.0",
        remote_interface_index: 517,
        local_interface_index: 505,
        type: 1,
        remote_system_name: "a7-mx80-2"
      },
      - {
        remote_interface_name: "eth1",
        local_interface_name: "ge-0/0/2.0",
        remote_interface_index: 1,
        local_interface_index: 507,
        type: 2,
        remote_system_name: "a7s35"
      },
      - {
        remote_interface_name: "eth1",
        local_interface_name: "ge-0/0/3.0",
        remote_interface_index: 1,
        local_interface_index: 509,
        type: 2,
        remote_system_name: "a7s36"
      }
    ]
  },
  - PRouterEntry: {
    + ipMib: [...],
    + ifTable: [...],
    + ifXTable: [...],
    + arpTable: [...],
    + lldpTable: [...],
    + ifStats: [...],
  }
}

```

5042435

Contrail User Interface for Underlay Overlay Analytics

The topology view and related functionality is accessed from the Contrail Web user interface, **Monitor > Physical Topology**.

Enabling Physical Topology on the Web UI

To enable the **Physical Topology** section in the Contrail Web UI:

1. Add the following lines to the `/etc/contrail/config.global.js` file of all the **contrail-webui** nodes:

```

config.optFeatureList = {};
config.optFeatureList.mon_infra_underlay = true;

```

2. Restart webui supervisor.

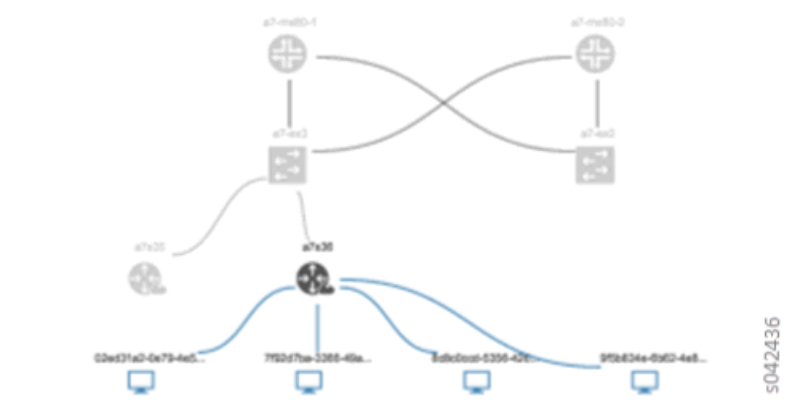
service supervisor-webui restart

The **Physical Topology** section is now available on the Contrail Web UI.

Viewing Topology to the Virtual Machine Level

In the Contrail user interface, it is possible to drill down through displayed topology to the virtual machine level. The following diagram shows the virtual machines instantiated on a7s36 vRouter and the full physical topology related to each.

Figure 177: Physical Topology Related to a vRouter



Viewing the Traffic of any Link

At **Monitor > Physical Topology**, double click any link on the topology to display the traffic statistics graph for that link. The following is an example.

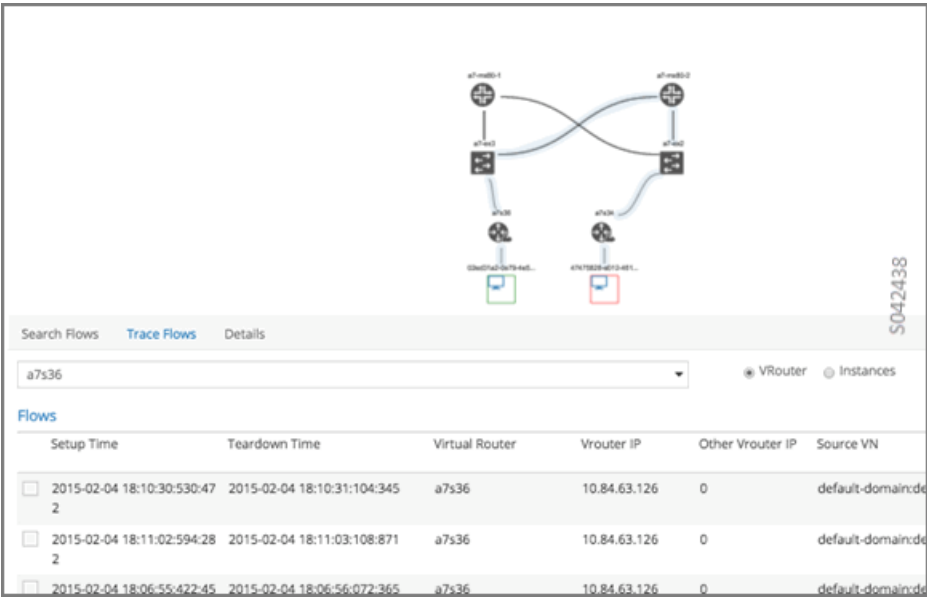
Figure 178: Traffic Statistics Graph



Trace Flows

Click the **Trace Flows** tab to see a list of active flows. To see the path of a flow, click a flow in the active flows list, then click the **Trace Flow** button. The path taken in the underlay by the selected flow displays. The following is an example.

Figure 179: List of Active Flows



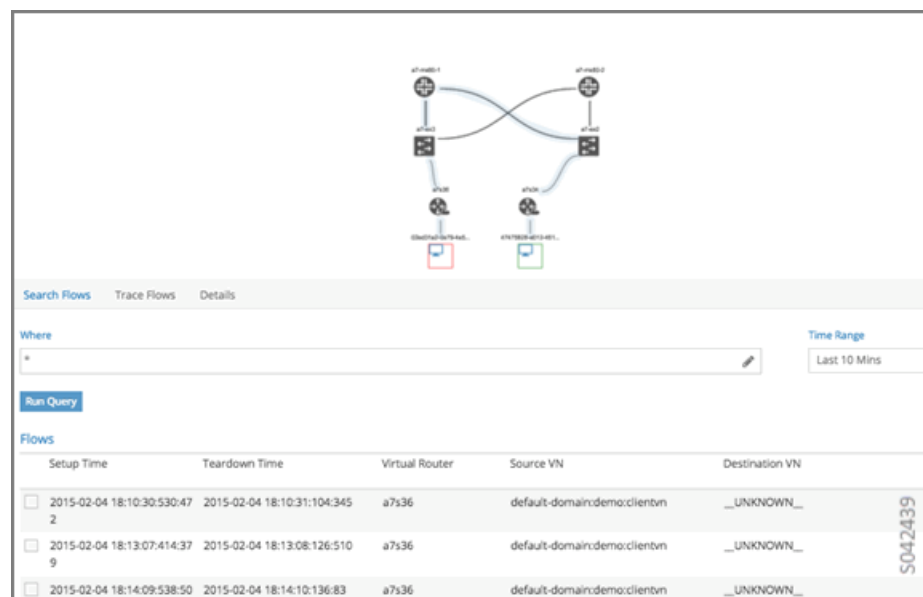
Limitations of Trace Flow Feature

Because the Trace Flow feature uses ip traceroute to determine the path between the two vRouters involved in the flow, it has the same limitations as the ip traceroute, including that Layer 2 routers in the path are not listed, and therefore do not appear in the topology.

Search Flows and Map Flows

Click the **Search Flows** tab to open a search dialog, then click the **Search** button to list the flows that match the search criteria. You can select a flow from the list and click **Map Flow** to display the underlay path taken by the selected flow in the topology. The following is an example.

Figure 180: Underlay Path



Overlay to Underlay Flow Map Schemas

The schema to query the underlay mapping information for an overlay flow is obtained from a REST API, which can be accessed on your system using a URL of the following form:

http://<host ip>:8081/analytics/table/OverlayToUnderlayFlowMap/schema

Example: Overlay to Underlay Flow Map Schema

```
{
  "type": "FLOW",
  "columns": [
    {
      "datatype": "string", "index": true, "name": "o_svn", "select": false,
      "suffixes": ["o_sip"]
    },
    {
      "datatype": "string", "index": false, "name": "o_sip", "select": false,
      "suffixes": null
    },
    {
      "datatype": "string", "index": true, "name": "o_dvn", "select": false,
      "suffixes": ["o_dip"]
    },
    {
      "datatype": "string", "index": false, "name": "o_dip", "select": false,

```

```

"suffixes": null},

{"datatype": "int", "index": false, "name": "o_sport", "select": false,
"suffixes": null},

{"datatype": "int", "index": false, "name": "o_dport", "select": false,
"suffixes": null},

{"datatype": "int", "index": true, "name": "o_protocol", "select": false,
"suffixes": ["o_sport", "o_dport"]},

{"datatype": "string", "index": true, "name": "o_vrouter", "select": false,
"suffixes": null},

{"datatype": "string", "index": false, "name": "u_prouter", "select": null,
"suffixes": null},

{"datatype": "int", "index": false, "name": "u_pifindex", "select": null,
"suffixes": null},

{"datatype": "int", "index": false, "name": "u_vlan", "select": null,
"suffixes": null},

{"datatype": "string", "index": false, "name": "u_sip", "select": null,
"suffixes": null},

{"datatype": "string", "index": false, "name": "u_dip", "select": null,
"suffixes": null},

{"datatype": "int", "index": false, "name": "u_sport", "select": null,
"suffixes": null},

{"datatype": "int", "index": false, "name": "u_dport", "select": null,
"suffixes": null},

{"datatype": "int", "index": false, "name": "u_protocol", "select": null,
"suffixes": null},

{"datatype": "string", "index": false, "name": "u_flowtype", "select": null,
"suffixes": null},

{"datatype": "string", "index": false, "name": "u_otherinfo", "select": null,
"suffixes": null}}}

```

The schema for underlay data across pRouters is defined in the Contrail installation at:

<http://<host ip>:8081/analytics/table/StatTable.UFlowData.flow/schema>

Example: Flow Data Schema for Underlay

```

{"type": "STAT",
"columns": [

{"datatype": "string", "index": true, "name": "Source", "suffixes": null},

{"datatype": "int", "index": false, "name": "T", "suffixes": null},

{"datatype": "int", "index": false, "name": "CLASS(T)", "suffixes": null},

{"datatype": "int", "index": false, "name": "T=", "suffixes": null},

```

```
{ "datatype": "int", "index": false, "name": "CLASS(T=)", "suffixes": null },
{ "datatype": "uuid", "index": false, "name": "UUID", "suffixes": null },
{ "datatype": "int", "index": false, "name": "COUNT(flow)", "suffixes": null },
{ "datatype": "string", "index": true, "name": "name", "suffixes":
  ["flow.pifindex"] },
{ "datatype": "int", "index": false, "name": "flow.pifindex", "suffixes": null },
{ "datatype": "int", "index": false, "name": "SUM(flow.pifindex)", "suffixes":
  null },
{ "datatype": "int", "index": false, "name": "CLASS(flow.pifindex)", "suffixes":
  null },
{ "datatype": "int", "index": false, "name": "flow.sport", "suffixes": null },
{ "datatype": "int", "index": false, "name": "SUM(flow.sport)", "suffixes":
  null },
{ "datatype": "int", "index": false, "name": "CLASS(flow.sport)", "suffixes":
  null },
{ "datatype": "int", "index": false, "name": "flow.dport", "suffixes": null },
{ "datatype": "int", "index": false, "name": "SUM(flow.dport)", "suffixes":
  null },
{ "datatype": "int", "index": false, "name": "CLASS(flow.dport)", "suffixes":
  null },
{ "datatype": "int", "index": true, "name": "flow.protocol", "suffixes":
  ["flow.sport", "flow.dport"] },
{ "datatype": "int", "index": false, "name": "SUM(flow.protocol)", "suffixes":
  null },
{ "datatype": "int", "index": false, "name": "CLASS(flow.protocol)", "suffixes":
  null },
{ "datatype": "string", "index": true, "name": "flow.sip", "suffixes": null },
{ "datatype": "string", "index": true, "name": "flow.dip", "suffixes": null },
{ "datatype": "string", "index": true, "name": "flow.vlan", "suffixes": null },
{ "datatype": "string", "index": false, "name": "flow.flowtype", "suffixes":
  null },
{ "datatype": "string", "index": false, "name": "flow.otherinfo", "suffixes":
  null } }
```

Example: Typical Query for Flow Map

The following is a typical query. Internally, the **analytics-api** performs a query into the **FlowRecordTable**, then into the **StatTable.UFlowData.flow**, to return list of (**prouter**, **pifindex**) pairs that give the underlay path taken for the given overlay flow.

```
FROM
  OverlayToUnderlayFlowMap

SELECT
  prouter, pifindex

WHERE
  o_svn, o_sip, o_dvn, o_dip, o_sport, o_dport, o_protocol = <overlay flow>
```

Module Operations for Overlay Underlay Mapping

SNMP Collector Operation

The Contrail SNMP collector uses a Net-SNMP library to talk to a physical router or any SNMP agent. Upon receiving SNMP packets, the data is translated to the Python dictionary, and corresponding UVE objects are created. The UVE objects are then posted to the SNMP collector.

The SNMP module sleeps for some configurable period, then forks a collector process and waits for the process to complete. The collector process goes through a list of devices to be queried. For each device, it forks a greenlet task (Python coroutine), accumulates SNMP data, writes the summary to a JSON file, and exits. The parent process then reads the JSON file, creates UVEs, sends the UVEs to the collector, then goes to sleep again.

The pRouter UVE sent by the SNMP collector carries only the raw MIB information.

Example: pRouter Entry Carried in pRouter UVE

The definition below shows the **pRouterEntry** carried in the **pRouterUVE**. Additionally, an example **LldpTable** definition is shown.

The following create a virtual table as defined by:

```
http://<host ip>:8081/analytics/table/StatTable.UFlowData.flow/schema

struct LldpTable {
  1: LldpLocalSystemData lldpLocalSystemData
  2: optional list<LldpRemoteSystemsData> lldpRemoteSystemsData
}

struct PRouterEntry {
  1: string name (key="ObjectPRouter")
  2: optional bool deleted
  3: optional LldpTable lldpTable
  4: optional list<ArpTable> arpTable
```

```
5: optional list<IfTable> ifTable
6: optional list<IfXTable> ifXTable
7: optional list<IfStats> ifStats (tags="name:.ifIndex")
8: optional list<IpMib> ipMib
}
uve sandesh PRouterUVE {
  1: PRouterEntry data
}
```

Topology Module Operation

The topology module reads UVEs posted by the SNMP collector and computes the neighbor table, populating the table with remote system name, local and remote interface names, the remote type (pRouter or vRouter) and local and remote ifindices. The topology module sleeps for a while, reads UVEs, then computes the neighbor table and posts the UVE to the collector.

The pRouter UVE sent by the topology module carries the neighbor list, so the clients can put together all of the pRouter neighbor lists to compute the full topology.

The corresponding pRouter UVE definition is the following.

```
struct LinkEntry {
  1: string remote_system_name
  2: string local_interface_name
  3: string remote_interface_name
  4: RemoteType type
  5: i32 local_interface_index
  6: i32 remote_interface_index
}
struct PRouterLinkEntry {
  1: string name (key="ObjectPRouter")
  2: optional bool deleted
  3: optional list<LinkEntry> link_table
}
uve sandesh PRouterLinkUVE {
  1: PRouterLinkEntry data
}
```

```
}
```

IPFIX and sFlow Collector Operation

An IPFIX and sFlow collector has been implemented in the Contrail collector. The collector receives the IPFIX and sFlow samples and stores them as statistics samples in the analytics database.

Example: IPFIX sFlow Collector Data

The following definition shows the data stored for the statistics samples and the indices that can be used to perform queries.

```
struct UFlowSample {
    1: u64 pifindex
    2: string sip
    3: string dip
    4: u16 sport
    5: u16 dport
    6: u16 protocol
    7: u16 vlan
    8: string flowtype
    9: string otherinfo
}

struct UFlowData {
    1: string name (key="ObjectPRouterIP")
    2: optional bool deleted
    3: optional list<UFlowSample> flow (tags="name:.pifindex, .sip, .dip,
        .protocol:.sport, .protocol:.dport, .vlan")
}
```

Troubleshooting Underlay Overlay Mapping

This section provides a variety of links where you can research errors that may occur with underlay overlay mapping.

System Logs Logs for **contrail-snmp-collector** and **contrail-topology** are in the following locations on an installed Contrail system:

`/var/log/contrail/contrail-snmp-collector-stdout.log`

`/var/log/contrail/contrail-topology.log`

Introspect Utility Use URLs of the following forms on your Contrail system to access the introspect utilities for SNMP data and for topology data.

- SNMP data introspect

`http://<host ip>:5920/Snh_SandeshUVECacheReq?x=PRouterEntry`

- Topology data introspect

`http://<host ip>:5921/Snh_SandeshUVECacheReq?x=PRouterLinkEntry`

Script to add pRouter Objects

The usual mechanism for adding pRouter objects to **contrail-config** is through Contrail UI. But you also have the ability to add these objects using the Contrail **vnc-api**. To add one pRouter, save the file with the name **cfg-snmp.py**, and then execute the command as shown:

python cfg-snmp.py

**Example: Content for
cfg-snmp.py**

```
#!/python

from vnc_api import vnc_api

from vnc_api.gen.resource_xsd import SNMPCredentials

vnc = vnc_api.VncApi('admin', 'abcde123', 'admin')
apr = vnc_api.gen.resource_client.PhysicalRouter(name='a7-mx80-1')
apr.set_physical_router_management_ip('ip_address')
apr.set_physical_router_dataplane_ip('ip_address')
apr.set_physical_router_snmp_credentials(SNMPCredentials(version=2,
v2_community='public'))

vnc.physical_router_create(apr)

#$ABC123

apr = vnc_api.gen.resource_client.PhysicalRouter(name='a7-mx80-2')
apr.set_physical_router_management_ip('ip_address')
apr.set_physical_router_dataplane_ip('ip_address')
apr.set_physical_router_snmp_credentials(SNMPCredentials(version=2,
v2_community='public'))
```



```
vnc.physical_router_create(apr)

#$ABC123'

apr = vnc_api.gen.resource_client.PhysicalRouter(name='a7-ex3')

apr.set_physical_router_management_ip('source_ip')

apr.set_physical_router_dataplane_ip('source_ip')

apr.set_physical_router_snmp_credentials(SNMPCredentials(version=2,
v2_community='public'))

vnc.physical_router_create(apr)

#$ABC123'

apr = vnc_api.gen.resource_client.PhysicalRouter(name='a7-ex2')

apr.set_physical_router_management_ip('ip_address')

apr.set_physical_router_dataplane_ip('ip_address')

apr.set_physical_router_snmp_credentials(SNMPCredentials(version=2,
v2_community='public'))

vnc.physical_router_create(apr)

#$ABC123'
```

- Related Documentation**
- [Understanding Contrail Analytics on page 515](#)
 - [Contrail Alerts on page 516](#)

CHAPTER 24

Configuring Contrail Analytics

- [Analytics Scalability on page 540](#)
- [High Availability for Analytics on page 541](#)
- [System Log Receiver in Contrail Analytics on page 542](#)
- [Sending Flow Messages to the Contrail System Log on page 543](#)
- [Ceilometer Support in a Contrail Cloud on page 544](#)
- [User Configuration for Analytics Alarms and Log Statistics on page 549](#)
- [Alarms History on page 557](#)
- [Node Memory and CPU Information on page 558](#)
- [Role- and Resource-Based Access Control for the Contrail Analytics API on page 559](#)
- [Configuring Analytics as a Standalone Solution on page 560](#)
- [Configuring Secure Sandesh and Introspect for Contrail Analytics on page 562](#)

Analytics Scalability

The Contrail monitoring and analytics services (*collector* role) collect and store data generated by various system components and provide the data to the Contrail interface by means of representational state transfer (REST) application program interface (API) queries.

The Contrail components are horizontally scalable to ensure consistent performance as the system grows. Scalability is provided for the generator components (*control* and *compute* roles) and for the REST API users (*webui* role).

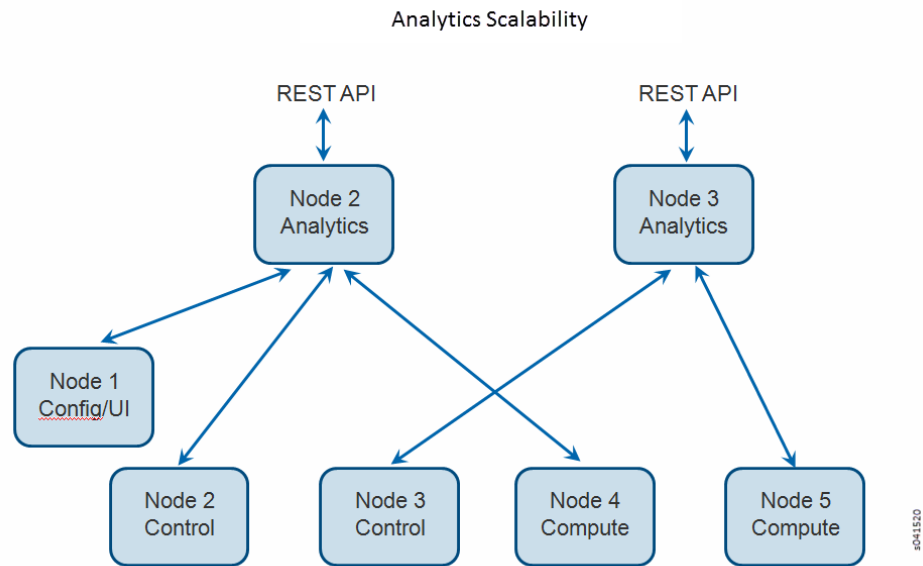
This section provides a brief description of the recommended configuration of analytics in Contrail to achieve horizontal scalability.

The following is the recommended locations for the various component roles of the Contrail system for a 5-node configuration.

- Node 1 —config role, web-ui role
- Node 2 —control role, analytics role, database role
- Node 3 —control role, analytics role, database role
- Node 4 —compute role
- Node 5 —compute role

[Figure 181 on page 541](#) illustrates scalable connections for analytics in a 5-node system, with the nodes configured for roles as recommended above. The analytics load is distributed between the two analytics nodes. This configuration can be extended to any number of analytics nodes.

Figure 181: Analytics Scalability



The analytics nodes collect and store data and provide this data through various REST API queries. Scalability is provided for the control nodes, the compute nodes, and the REST API users, with the API output displayed in the Contrail user interface. As the number of control and compute nodes increase in the system, the analytics nodes can also be increased.

High Availability for Analytics

Contrail supports multiple instances of analytics for high availability and load balancing.

Contrail analytics provides two broad areas of functionality:

- **contrail-collector** —Receives status, logs, and flow information from all Contrail processing elements (for example, generators) and records them.

Every generator is connected to one of the **contrail-collector** instances at any given time. If an instance fails (or is shut down), all the generators that are connected to it are automatically moved to another functioning instance, typically in a few seconds or less. Some messages may be lost during this movement. UVEs are resilient to message loss, so the state shown in a UVE is kept consistent to the state in the generator.

- **contrail-opserver** —Provides an external API to report UVEs and to query logs and flows.

Each analytics component exposes a northbound REST API represented by the **contrail-opserver** service (port 8081) so that the failure of one analytics component or one **contrail-opserver** service should not impact the operation of other instances.

These are the ways to manage connectivity to the **contrail-opserver** endpoints:

- Periodically poll the **contrail-opserver** service on a set of analytics nodes to determine the list of functioning endpoints, then make API requests from one or more of the functioning endpoints.
- The Contrail user interface makes use of the same northbound REST API to present dashboards, and reacts to any **contrail-opserver** high availability event automatically.

System Log Receiver in Contrail Analytics

- [Overview on page 542](#)
- [Redirecting System Logs to Contrail Collector on page 542](#)
- [Exporting Logs from Contrail Analytics on page 542](#)

Overview

The **contrail-collector** process on the Contrail Analytics node can act as a system log receiver.

Redirecting System Logs to Contrail Collector

You can enable the **contrail-collector** to receive system logs by giving a valid **syslog_port** as a command line option:

```
--DEFAULT.syslog_port <arg>
```

or by adding **syslog_port** in the **DEFAULT** section of the configuration file at **/etc/contrail/contrail-collector.conf**.

For nodes to send system logs to the **contrail-collector**, the system log configuration for the node should be set up to direct the system logs to **contrail-collector**.

Example Add the following line in **/etc/rsyslog.d/50-default.conf** on an Ubuntu system to redirect the system logs to **contrail-collector**.

```
*.* @<collector_ip>:<collector_syslog_port> :: @ for udp, @@ for tcp
```

The logs can be retrieved by using Contrail tool, either by using the **contrail-logs** utility on the analytics node or by using the Contrail user interface on the system log query page.

Exporting Logs from Contrail Analytics

You can also export logs stored in Contrail analytics to another system log receiver by using the **contrail-logs** utility.

The **contrail-logs** utility can take these options: **--send-syslog**, **--syslog-server**, **--syslog-port**, to query Contrail analytics, then send the results as system logs to a system log server. This is an on-demand command, one can write a cron job or a job that continuously invokes **contrail-logs** to achieve continuous sending of logs to another system log server.

Sending Flow Messages to the Contrail System Log

The **contrail-vrouter-agent** can be configured to send flow messages and other messages to the system log (syslog). To send flow messages to syslog, configure the following parameters in **/etc/contrail/contrail-vrouter-agent.conf**.

The following parameters are under the section **DEFAULT**:

- **log_flow=1**—Enables logging of all flow messages.
- **use_syslog=1**—Enables sending of all messages, including flow messages, to syslog.
- **syslog_facility=LOG_LOCAL0**—Enables sending messages from the **contrail-vrouter-agent** to the syslog, using the facility **LOCAL0**. You can configure **LOCAL0** to your required facility.
- **log_level=SYS_INFO**—Changes the logging level of **contrail-vrouter-agent** to INFO.

If syslog is enabled, flow messages are *not* sent to Contrail Analytics because the two destinations are mutually exclusive.

Flow log sampling settings apply regardless of the flow log destination specified. If sampling is enabled, the syslog messages will be sampled using the same rules that would apply to Contrail Analytics. If non-sampled flow data is required, sampling must be disabled by means of configuration settings.

Flow events for termination will include both the appropriate tear-down fields and the appropriate setup fields.

The flow messages will be sent to the syslog with a severity of INFO.

The user can configure the remote system log (**rsyslog**) on the compute node to send syslog messages with facility LOCAL0, severity of INFO (and lower), to the remote syslog server. Messages with a higher severity than INFO can be logged to a local file to allow for debugging.

Flow messages appear in the syslog in a format similar to the following log example:

```
May 24 14:40:13 a7s10 contrail-vrouter-agent[29930]: 2016-05-24 Tue 14:40:13:921.098
PDT a7s10 [Thread 139724471654144, Pid 29930]: [SYS_INFO]: FlowLogDataObject:
flowdata= [ [ [ flowuuid = 7ea8bf8f-b827-496e-b93e-7622a0c8eeea direction_ing = 1
sourcevn = default-domain:mock-gen-test:vn8 sourceip = 1.0.0.9 destvn =
default-domain:mock-gen-test:vn58 destip = 1.0.0.59 protocol = 1 sport = -29520 dport =
20315 setup_time = 1464125225556930 bytes = 1035611592 packets = 2024830 diff_bytes
= 27240 diff_packets = 40 ], ] ]
```



NOTE: Several individual flow messages might be packed into a single syslog message for improved efficiency.

Ceilometer Support in a Contrail Cloud

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

- [Overview on page 544](#)
- [Ceilometer Details on page 544](#)
- [Verification of Ceilometer Operation on page 545](#)
- [Contrail Ceilometer Plugin on page 547](#)
- [Ceilometer Installation and Provisioning on page 549](#)

Overview

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

The prerequisites for installing Ceilometer are:

- Contrail Cloud installation
- Provisioned using Server Manager Lite with `enable_ceilometer = True` in the `testbed.py` file.
- Alternately, provisioned using Server Manager with `enable_ceilometer = True` in the `cluster.json` or `cluster` configuration.



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

Ceilometer Details

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

The Ceilometer architecture consists of:

Polling agent—Agent designed to poll OpenStack services and build meters. The polling agents are also run on the compute nodes in addition to the OpenStack controller.

Notification agent—Agent designed to listen to notifications on message queue and convert them to events and samples.

Collector—Gathers and records event and metering data created by the notification and polling agents.

API server—Provides a REST API to query and view data recorded by the collector service.

Alarms—Daemons to evaluate and notify based on defined alarming rules.

Database—Stores the metering data, notifications, and alarms. The supported databases are MongoDB, SQL-based databases compatible with SQLAlchemy, and HBase. The recommended database is MongoDB, which has been thoroughly tested with Contrail and deployed on a production scale.

Verification of Ceilometer Operation

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

On Ubuntu, the service names are:

Polling agent—ceilometer-agent-central and ceilometer-agent-compute

Notification agent—ceilometer-agent-notification

Collector—ceilometer-collector

API Server—ceilometer-api

Alarms—ceilometer-alarm-evaluator and ceilometer-alarm-notifier

On RHEL Server 7.0, the service names are:

Polling agent—openstack-ceilometer-central and openstack-ceilometer-compute

Notification agent—openstack-ceilometer-notification

Collector—openstack-ceilometer-collector

API server—openstack-ceilometer-api

Alarms—openstack-ceilometer-alarm-evaluator and openstack-ceilometer-alarm-notifier

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

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

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

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

```
user@host:~# (source /etc/contrail/openstackrc; ceilometer meter-list)
```

Name	User ID	Type	Unit	Resource ID	Project ID
ip.floating.receive.bytes	a726f93a-65fa-4cad-828b-54dbfcf4a119	cumulative	B		None
ip.floating.receive.packets	a726f93a-65fa-4cad-828b-54dbfcf4a119	cumulative	packet		None
ip.floating.transmit.bytes	a726f93a-65fa-4cad-828b-54dbfcf4a119	cumulative	B		None
ip.floating.transmit.packets	a726f93a-65fa-4cad-828b-54dbfcf4a119	cumulative	packet		None
network	7fa6796b-756e-4320-9e73-87d4c52ecc83	gauge	network	15c0240142084d16b3127d6f844adb9	
network	9408e287-d3e7-41e2-89f0-5c691c9ca450	gauge	network	15c0240142084d16b3127d6f844adb9	
network	b3b72b98-f61e-4e1f-9a9b-84f4f3ddec0b	gauge	network	15c0240142084d16b3127d6f844adb9	
network	cb829abd-e6a3-42e9-a82f-0742db55d329	gauge	network	15c0240142084d16b3127d6f844adb9	
network.create	7fa6796b-756e-4320-9e73-87d4c52ecc83	delta	network	15c0240142084d16b3127d6f844adb9	
network.create	9408e287-d3e7-41e2-89f0-5c691c9ca450	delta	network	15c0240142084d16b3127d6f844adb9	
network.create	b3b72b98-f61e-4e1f-9a9b-84f4f3ddec0b	delta	network	15c0240142084d16b3127d6f844adb9	
network.create	cb829abd-e6a3-42e9-a82f-0742db55d329	delta	network	15c0240142084d16b3127d6f844adb9	
port	0d401d96-c2bf-4672-abf2-880eecf25ceb	gauge	port	01edcedd989f43b3a2d6121d424b254d	
port	211b94a4-581d-45d0-8710-c6c69df15709	gauge	port	01edcedd989f43b3a2d6121d424b254d	
port	2287ce25-4eef-4212-b77f-3cf590943d36	gauge	port	01edcedd989f43b3a2d6121d424b254d	
port.create	f62f3732-222e-4c40-8783-5bcbcf1fd6a1c	delta	port	01edcedd989f43b3a2d6121d424b254d	
port.create	f8c89218-3cad-48e2-8bd8-46c1bc33e752	delta	port	01edcedd989f43b3a2d6121d424b254d	

```

| port.update | delta | port |
43ed422d-b073-489f-877f-515a3cc0b8c4 | 15c0240142084d16b3127d6f844adbd9 |
ded208991de34fe4bb7dd725097f1c7e |
| subnet | gauge | subnet |
09105ed1-1654-4b5f-8c12-f0f2666fa304 | 15c0240142084d16b3127d6f844adbd9 |
ded208991de34fe4bb7dd725097f1c7e |
| subnet | gauge | subnet |
4bf00aac-407c-4266-a048-6ff52721ad82 | 15c0240142084d16b3127d6f844adbd9 |
ded208991de34fe4bb7dd725097f1c7e |
| subnet.create | delta | subnet |
09105ed1-1654-4b5f-8c12-f0f2666fa304 | 15c0240142084d16b3127d6f844adbd9 |
ded208991de34fe4bb7dd725097f1c7e |
| subnet.create | delta | subnet |
4bf00aac-407c-4266-a048-6ff52721ad82 | 15c0240142084d16b3127d6f844adbd9 |
ded208991de34fe4bb7dd725097f1c7e |

```



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

Contrail Ceilometer Plugin

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

```

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

```

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

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

```

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

```

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

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

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

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

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

```
| user_id      | None
|
+-----+
```

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

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

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

Ceilometer Installation and Provisioning

There are two scenarios possible for Contrail Ceilometer plugin installation.

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

User Configuration for Analytics Alarms and Log Statistics

- [Configuring Alarms Based on User-Visible Entities Data on page 549](#)
- [Examples: Detecting Anomalies on page 551](#)
- [Configuring the User-Defined Log Statistic on page 552](#)
- [Implementing the User-Defined Log Statistic on page 555](#)

Configuring Alarms Based on User-Visible Entities Data

Starting with Contrail 3.1, users can dynamically configure alarms based on the user-visible entities (UVE) data. An alarm configuration object is created based on the alarm configuration XSD schema. The alarm configuration object is added to the Contrail configuration database, using the Contrail API server REST API interface.

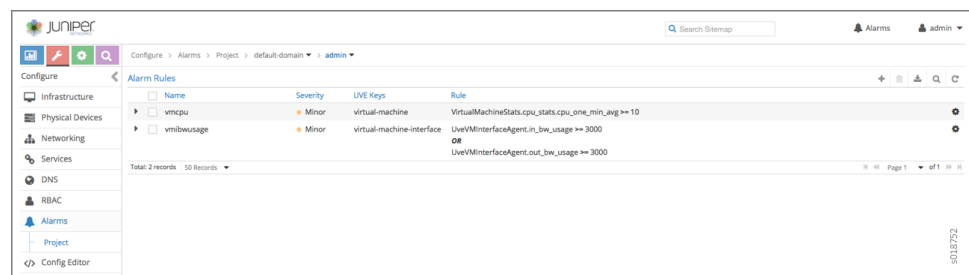
The schema for alarm configuration can be found at:

<https://github.com/Juniper/contrail-controller/blob/master/src/schema/alarm.xsd>

An alarm configuration object can be anchored in the configuration data model under **global-system-config** or **project**, depending on the alarm type. Under **global-system-config**, you should configure virtual network system-wide alarms, such as those for the analytics node, the config node, and so on. Under **project**, you should configure alarms related to project objects, such as virtual networks and similar objects.

To configure and monitor alarms using the Contrail UI:

1. Navigate to **Configure > Alarms > Project**, and select the desired project to access the **Alarm Rules** page.



2. Click the Gear icon to add a new alarm configuration or to edit an existing alarm configuration. Use the **Edit** screen to define descriptions and to set up alarm rules. See [Table 56 on page 550](#) for field descriptions.

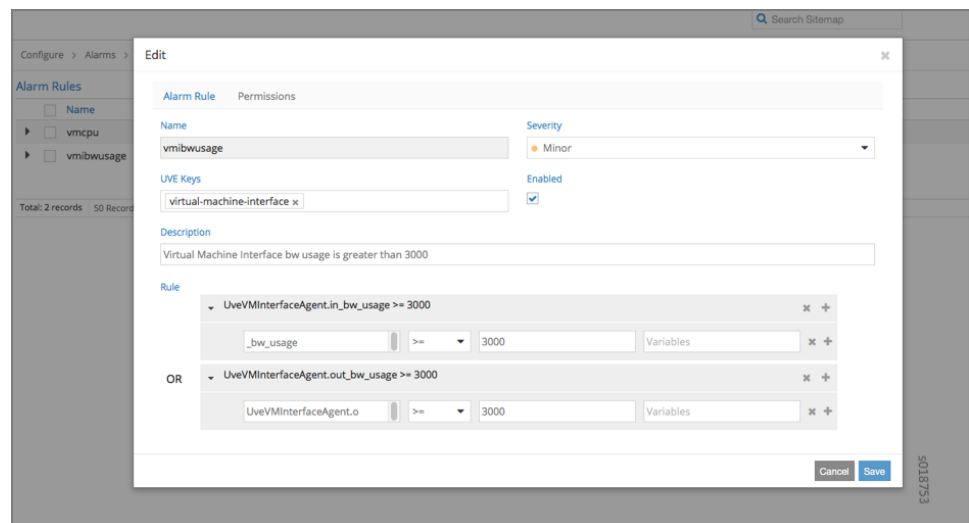


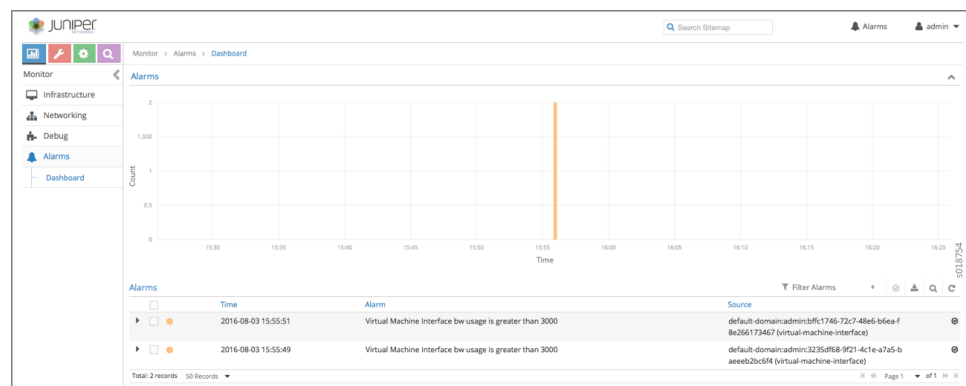
Table 56: Alarm Rules Fields

Field	Description
Name	Enter a name for the alarm.

Table 56: Alarm Rules Fields (*continued*)

Field	Description
Severity	Select the severity level of the alarm from the list.
UVE Keys	Select the list of UVE types to apply to this alarm.
Description	Enter a description of the alarm.
Rule	Set up the alarm rules. Alarm rules are expressed as OR of AND terms. Each term has operand1, operand2, and the operation. Operand1 is the UVE attribute. Operand2 can be either another UVE attribute or a JSON value. The rules are evaluated in the contrail-alarm-gen service and an alarm is raised or cleared as needed on respective conditions.

- To monitor alarms, navigate to **Monitor > Alarms > Dashboard**. The **Dashboard** screen lists the active alarms in the system.



Examples: Detecting Anomalies

The purpose of anomaly detection in Contrail is to identify a condition in which a metric deviates from its expected value, within given parameters.

Contrail uses a statistical process control model for time-series anomaly detection that can be computed online, in real-time. Raw metrics are sent as statistics by Sandesh generators embedded inside the UVEs. The model uses the running average and running standard deviation for a given raw metric. The model does not account for seasonality and linear trends in the metric.

The following example represents part of the UVE sent by the vRouter to the collector. The raw metrics are **phy_band_in_bps** and **phy_band_out_bps**.

The derived statistics are in **in_bps_ewm** and **out_bps_ewm**, which are generated when the model's EWM algorithm is applied to the raw metrics. The raw metrics and the derived statistics are part of the UVE and are sent to the collector.

```
struct EWMResult {
    3: u64 samples
```

```

        6: double mean
        7: double stddev
    }

    struct VrouterStatsAgent { // Agent stats

    1: string name (key="ObjectVRouter")

    2: optional bool deleted    ...

    /** @display_name:Vrouter Physical Interface Input bandwidth Statistics*/

    50: optional map<string,u64> phy_band_in_bps (tags="name:.__key")

    /** @display_name:Vrouter Physical Interface Output bandwidth Statistics*/

    51: optional map<string,u64> phy_band_out_bps (tags="name:.__key")

    52: optional map<string,derived_stats_results.EWMResult> in_bps_ewm
        (mstats="phy_band_in_bps:DSEWM:0.2")

    53: optional map<string,derived_stats_results.EWMResult> out_bps_ewm
        (mstats="phy_band_out_bps:DSEWM:0.2")
    }

```

The following shows part of the UVE that lists the raw metric **phy_band_out_bps** and the derived statistic **out_bps_ewm**. The user can define an alarm based on the values in **sigma** or in **stddev**.

```

- out_bps_ewm: {
  - eth0: {
    sigma: -0.425095,
    samples: 177,
    stddev: 6348.16,
    mean: 206712
  },
- phy_band_out_bps: {
  eth0: "204013"
},

```

s018755

Configuring the User-Defined Log Statistic

Any deployment of Contrail cloud over an orchestration system requires tools for monitoring and troubleshooting the entire cloud deployment. Cloud data centers are built with a large collection of interconnected servers that provide computing and storage capacity for a variety of applications. The monitoring of the cloud and its infrastructure requires monitoring logs and messages sent to a variety of servers from many micro services.

Contrail analytics stores all of the monitored messages in the Contrail database node, and the analytics generates a large amount of useful information that aids in monitoring and troubleshooting the network.

Starting with Contrail Release 3.1, the user-defined log statistic feature provides additional abilities for monitoring and troubleshooting by enabling the user to set a counter on any regular Perl-type expression. Each time the pattern is found in any system logs, UVEs, or object logs, the counter is incremented.

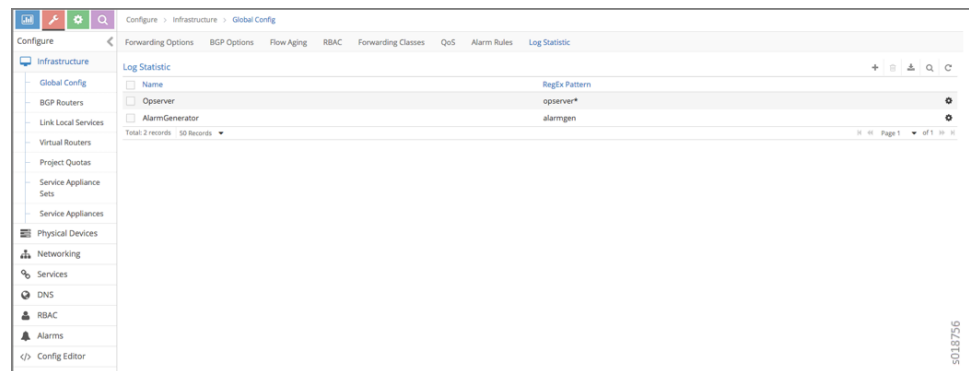
The XSD schema for the user-defined log statistic is located at:

https://github.com/Juniper/contrail-controller/blob/master/src/schema/usr_def_cntr.xsd

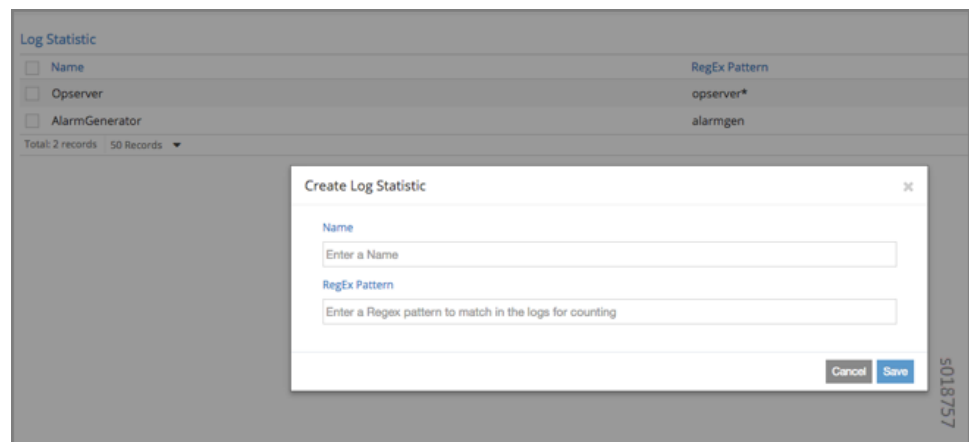
The user-defined log statistic can be configured from the Contrail UI or from the command line, using `vnc_api`.

To configure the user-defined log statistic from the Contrail UI:

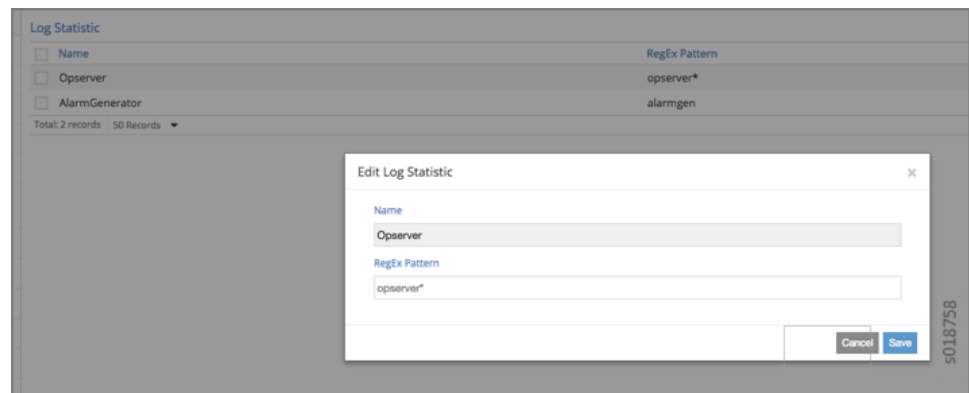
1. Navigate to **Configure > Infrastructure > Global Config** and select **Log Statistic**.



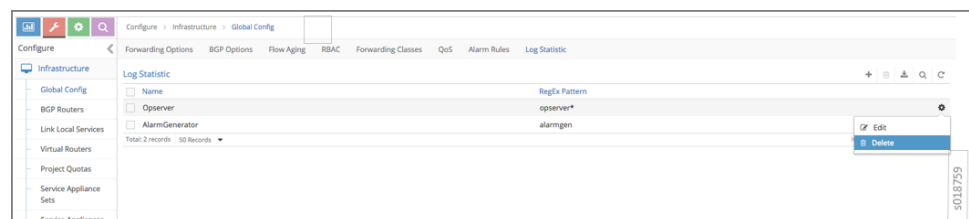
2. To create a log statistic, click the plus (+) icon to access the **Create Log Statistic** screen. Enter a name for the user-defined log statistic, and in the **RegExp Pattern** field, enter the Perl-type expression to look for and count.



- To edit an existing log statistic, select the name of the statistic and click the Gear icon, then select **Edit** to access the **Edit Log Statistic** screen.



- To delete a log statistic, select the name of the statistic and click the gear icon, then select the **Delete** option.



To configure the user-defined statistic from the **vnc_api**:

```
user@host:~# python
Python 2.7.6 (default, Jun 22 2015, 17:58:13)
[GCC 4.8.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
```

```
>> from vnc_api import vnc_api
>> from vnc_api.gen.resource_xsd import UserDefinedLogStat
>> from vnc_api.gen.resource_client import GlobalSystemConfig
>> vnc = vnc_api.VncApi('<username>', '<password>', '<tenant>')
>> gsc_uuid =
vnc.global_system_configs_list()['global-system-configs'][0]['uuid']
>> gsc = vnc.global_system_config_read(id=gsc_uuid)
```

To list the counters:

```
>> [(x.name, x.pattern) for x in gsc.user_defined_log_statistics.statlist]

[('HostnameCounter', 'dummy'), ('MyIp', '10.84.14.38')]
```

To add a counter:

```
>> g=GlobalSystemConfig()
>> g.add_user_defined_counter(UserDefinedLogStat('Foo', 'Ba.*r'))
>> vnc.global_system_config_update(g)
```

To verify an addition:

```
>> gsc = vnc.global_system_config_read(id=gsc_uuid)
>> [(x.name, x.pattern) for x in gsc.user_defined_log_statistics.statlist]

[('HostnameCounter', 'dummy'), ('MyIp', '10.84.14.38'), ('Foo', 'Ba.*r')]
```

Implementing the User-Defined Log Statistic

The statistics are sent as a counter that has been aggregated over a time period of 60 seconds.

A current sample from your system can be obtained from the UVE at:

`http://<analytics-ip>:8081/analytics/uves/user-defined-log-statistic/<name>`

You can also use the statistics table **UserDefinedLogStatTable** to get historical data with all supported aggregations such as SUM, AVG, and the like.

The schema for the table is at the following location:

`http://<ip>:8081/analytics/table/StatTable.UserDefinedCounter.count/schema`

Schema for User-Defined Statistics Table

The following is the schema for the user-defined statistic table:

```
{
  "type": "STAT",
  "columns": [
    {
      "datatype": "string",
      "index": true,
      "name": "Source",
      "suffixes": null
    },
    {
      "datatype": "int",
      "index": false,
      "name": "T",
      "suffixes": null
    },
    {
      "datatype": "int",
      "index": false,
      "name": "CLASS(T)",
      "suffixes": null
    },
    {
      "datatype": "int",
      "index": false,
      "name": "T=",
      "suffixes": null
    },
    {
      "datatype": "int",
      "index": false,
      "name": "CLASS(T=)",
      "suffixes": null
    }
  ],
}
```

```
{
  "datatype": "uuid",
  "index": false,
  "name": "UUID",
  "suffixes": null
},
{
  "datatype": "int",
  "index": false,
  "name": "COUNT(count)",
  "suffixes": null
},
{
  "datatype": "int",
  "index": false,
  "name": "count.previous",
  "suffixes": null
},
{
  "datatype": "int",
  "index": false,
  "name": "SUM(count.previous)",
  "suffixes": null
},
{
  "datatype": "int",
  "index": false,
  "name": "CLASS(count.previous)",
  "suffixes": null
},
{
  "datatype": "int",
  "index": false,
  "name": "MAX(count.previous)",
  "suffixes": null
},
{
  "datatype": "int",
  "index": false,
  "name": "MIN(count.previous)",
  "suffixes": null
},
{
  "datatype": "percentiles",
  "index": false,
  "name": "PERCENTILES(count.previous)",
  "suffixes": null
},
{
  "datatype": "avg",
  "index": false,
  "name": "AVG(count.previous)",
  "suffixes": null
},
{
  "datatype": "string",
  "index": true,
  "name": "name",
  "suffixes": null
}
}
```

```

    }
}

```

Alarms History

Starting with Contrail Release 4.0, you can view a history of alarms that were raised or reset. You can also view a history of user-visible entities (UVEs) that have been changed.

- [Viewing Alarms History on page 557](#)

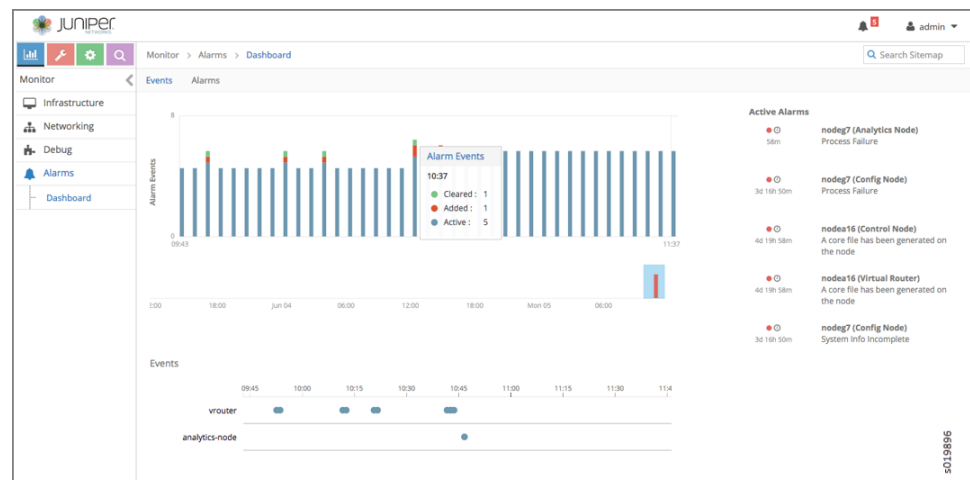
Viewing Alarms History

In the Contrail Web user interface, new fields at **Monitor > Alarms > Dashboard > Alarms History** now display alarms history, including alarms that were set or reset.

[Figure 182 on page 557](#) shows the alarms history, identifying the volume and types of alarms by time and the node types in which events are occurring. The right side panel lists by name the nodes in which active events are occurring.

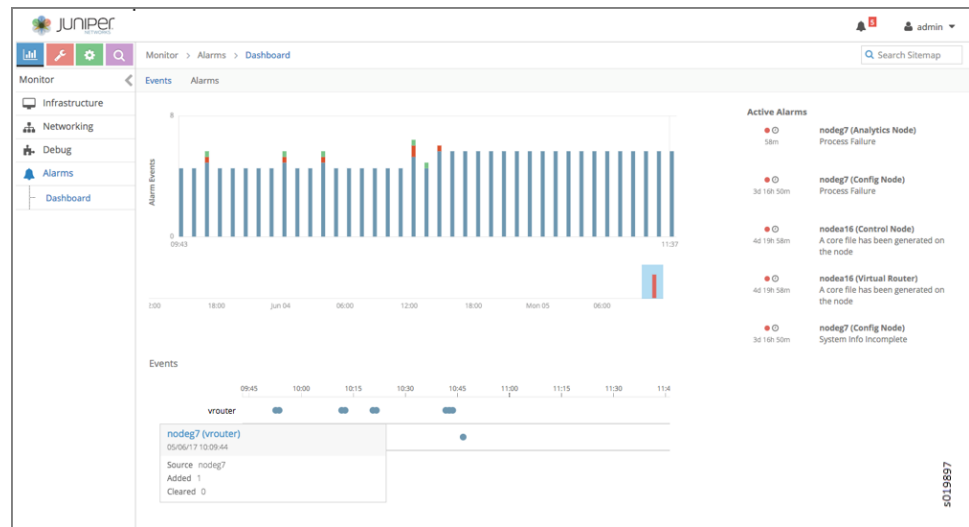
You can also use a **contrail-status** query to view the alarms history. Additionally, the **contrail-status** displays a history of added, updated, and removed information for UVEs in Contrail.

Figure 182: Alarms History Page



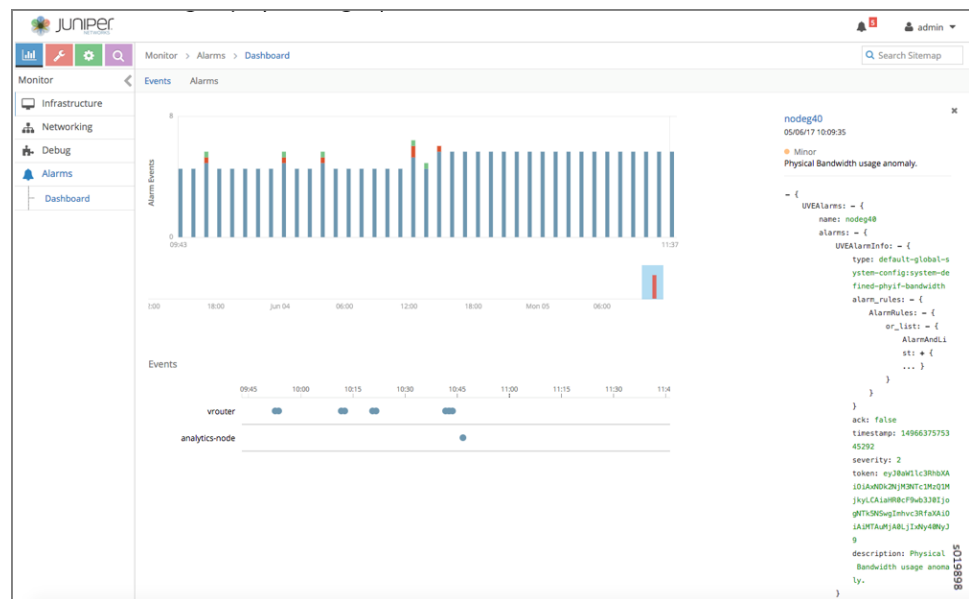
Tooltips are available on the Alarms History page. In the Events area, you can click on any node type listed to display a tooltip showing details of the events that have been added and cleared in that node, see [Figure 183 on page 558](#).

Figure 183: Events Log Tooltip



You can expand the event log in the right side panel to display a detailed event log. Click the name of any node in the list in the right panel, and the details of the current alarms are visible in the expanded view, see [Figure 184 on page 558](#).

Figure 184: Detailed Event Log



Related Documentation

- [User Configuration for Analytics Alarms and Log Statistics on page 549](#)

Node Memory and CPU Information

To help in monitoring and debugging, the following statistics have been added for all node types. The statistics are updated every 60 seconds.

- System CPU info
- System memory and CPU usage
- Memory and CPU usage of all processes

You can see a current sample from the UVE in your system at:

`http://<analytics-ip>:8081/analytics/uves/<node-type>/<hostname>?flat`

You can also use the statistics tables to get historical data with all supported aggregations, such as SUM, AVG, and so on:

- `NodeStatus.process_mem_cpu_usage`
- `NodeStatus.system_mem_cpu_usage`

The schema for the tables are at the following locations on your system:

`http://<analytics-ip>:8081/analytics/table/StatTable.NodeStatus.process_mem_cpu_usage/schema`

`http://<analytics-ip>:8081/analytics/table/StatTable.NodeStatus.system_mem_cpu_usage/schema`

Related Documentation

- [User Configuration for Analytics Alarms and Log Statistics on page 549](#)

Role- and Resource-Based Access Control for the Contrail Analytics API

In previous releases of Contrail, any user can access the Contrail analytics API by using queries to get historical information and by using UVEs to get state information.

Starting with Contrail Release 3.1, it is possible to restrict access such that only the **cloud-admin** user can access the Contrail analytics API.

Implementation details are as follows:

- An external user makes a REST API call to **contrail-analytics-api**, passing a token representing the user with the HTTP header `X-Auth-Token`.
- Based on the user role, **contrail-analytics-api** will only allow access for the **cloud-admin** user and reject the request (**HTTPUnauthorized**) for other users.

To set the **cloud_admin** user, use the following fields in `/etc/contrail/contrail-analytics-api.conf`:

- **aaa_mode**—Takes one of these values:
 - **no-auth**
 - **cloud-admin**
- **cloud_admin_role**—The user with this role has full access to everything. By default, this is set to "admin". This role must be configured in Keystone.

**Related
Documentation**

- [User Configuration for Analytics Alarms and Log Statistics on page 549](#)

Configuring Analytics as a Standalone Solution

Starting with Contrail 4.0, it is possible to configure Contrail Analytics as a standalone solution.

- [Overview: Contrail Analytics as a Standalone Solution on page 560](#)
- [Configuration Examples for Standalone on page 560](#)

Overview: Contrail Analytics as a Standalone Solution

Starting with Contrail 4.0 (containerized Contrail), Contrail Analytics can be configured as a standalone solution.

The following services are necessary for a standalone solution:

- config
- webui
- analytics
- analyticsdb

A standalone Contrail Analytics solution consists of the following containers:

- controller container with only config and webui services enabled
- analytics container
- analyticsdb container

Configuration Examples for Standalone

The following are examples of default inventory file configurations for the controller container for standalone Contrail analytics.

- [Examples: Inventory File Controller Components on page 560](#)
- [JSON Configuration Examples on page 561](#)

Examples: Inventory File Controller Components

The following are example analytics standalone solution inventory file configurations for Contrail controller container components.

- [Single Node Cluster on page 560](#)
- [Multi-Node Cluster on page 561](#)

Single Node Cluster

```
[contrail-controllers]
10.xx.32.10      controller_components=['config','webui']
```



```
[contrail-analyticsdb]
10.xx.32.10
```

```
[contrail-analytics]
10.xx.32.10
```

Multi-Node Cluster

```
[contrail-controllers]
10.xx.32.10      controller_components=['config','webui']
10.xx.32.11      controller_components=['config','webui']
10.xx.32.12      controller_components=['config','webui']
```

```
[contrail-analyticsdb]
10.xx.32.10
10.xx.32.11
10.xx.32.12
```

```
[contrail-analytics]
10.xx.32.10
10.xx.32.11
10.xx.32.12
```

JSON Configuration Examples

The following are example JSON file configurations for (**server.json**) for Contrail analytics standalone solution.

- [Example: JSON Single Node Cluster on page 561](#)
- [Example: JSON Multi-Node Cluster on page 561](#)

Example: JSON Single Node Cluster

```
{
  "cluster_id": "cluster1",
  "domain": "sm-domain.com",
  "id": "server1",
  "parameters": {
    "provision": {
      "contrail_4": {
        "controller_components": "['config','webui']"
      },
      ...
    }
  }
}
```

Example: JSON Multi-Node Cluster

```
{
  "cluster_id": "cluster1",
  "domain": "sm-domain.com",
  "id": "server1",
  "parameters": {
    "provision": {
      "contrail_4": {
        "controller_components": "['config','webui']"
      },
      ...
    },
    ...
  }
}
```

```

        "cluster_id": "cluster1",
        "domain": "sm-domain.com",
        "id": "server2",
        "parameters" : {
            "provision": {
                "contrail_4": {
                    "controller_components": "['config','webui']"
                },
                ...
            },
            {
                "cluster_id": "cluster1",
                "domain": "sm-domain.com",
                "id": "server3",
                "parameters" : {
                    "provision": {
                        "contrail_4": {
                            "controller_components": "['config','webui']"
                        },
                        ...
                    },
                    ...
                }
            }
        }

```

- Related Documentation**
- [Configuring Secure Sandesh and Introspect for Contrail Analytics on page 562](#)
 - [Understanding Contrail Analytics on page 515](#)

Configuring Secure Sandesh and Introspect for Contrail Analytics

- [Configuring Secure Sandesh Connection on page 562](#)
- [Configuring Secure Introspect Connection on page 563](#)

Configuring Secure Sandesh Connection

All Contrail services use Sandesh, a southbound interface protocol based on Apache Thrift, to send analytics data such as system logs, object logs, UVEs, flow logs, and the like, to the collector service in the Contrail Analytics node. The Transport Layer Security (TLS) protocol is used for certificate exchange, mutual authentication, and negotiating ciphers to secure the Sandesh connection from potential tampering and eavesdropping.

To configure a secure Sandesh connection, configure the following parameters in all Contrail services that connect to the collector (Sandesh clients) and the Sandesh server.

Parameter	Description	Default
[SANDESH].sandesh_keyfile	Path to the node's private key	/etc/contrail/ssl/private/server-privkey.pem
[SANDESH].sandesh_certfile	Path to the node's public certificate	/etc/contrail/ssl/certs/server.pem
[SANDESH].sandesh_ca_cert	Path to the CA certificate	/etc/contrail/ssl/certs/ca-cert.pem

Parameter	Description	Default
[SANDESH].sandesh_ssl_enable	Enable or disable secure Sandesh connection	false

Configuring Secure Introspect Connection

All Contrail services are embedded with a web server that can be used to query the internal state of the data structures, view trace messages, and perform other extensive debugging. The Transport Layer Security (TLS) protocol is used for certificate exchange, mutual authentication, and negotiating ciphers to secure the introspect connection from potential tampering and eavesdropping.

To configure a secure introspect connection, configure the following parameters in the Contrail service, see [Table 57 on page 563](#).

Table 57: Secure Introspect Parameters

Parameter	Description	Default
[SANDESH].sandesh_keyfile	Path to the node's private key	/etc/contrail/ssl/private/server-privkey.pem
[SANDESH].sandesh_certfile	Path to the node's public certificate	/etc/contrail/ssl/certs/server.pem
[SANDESH].sandesh_ca_cert	Path to the CA certificate	/etc/contrail/ssl/certs/ca-cert.pem
[SANDESH].introspect_ssl_enable	Enable or disable secure introspect connection	false

CHAPTER 25

Using Contrail Analytics to Monitor and Troubleshoot the Network

- [Monitoring the System on page 566](#)
- [Debugging Processes Using the Contrail Introspect Feature on page 568](#)
- [Monitor > Infrastructure > Dashboard on page 572](#)
- [Monitor > Infrastructure > Control Nodes on page 574](#)
- [Monitor > Infrastructure > Virtual Routers on page 581](#)
- [Monitor > Infrastructure > Analytics Nodes on page 591](#)
- [Monitor > Infrastructure > Config Nodes on page 596](#)
- [Monitor > Networking on page 599](#)
- [Query > Flows on page 607](#)
- [Query > Logs on page 614](#)
- [Understanding Flow Sampling on page 619](#)
- [Example: Debugging Connectivity Using Monitoring for Troubleshooting on page 622](#)

Monitoring the System

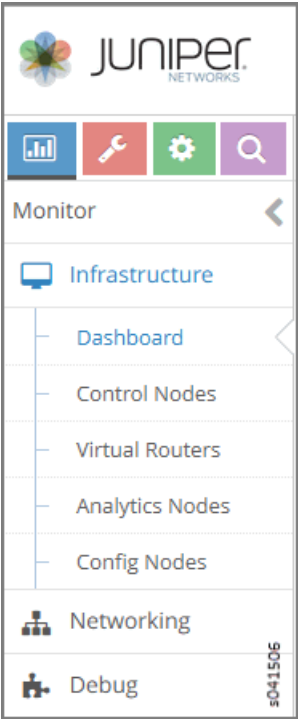
The **Monitor** icon on the Contrail Controller provides numerous options so you can view and analyze usage and other activity associated with all nodes of the system, through the use of reports, charts, and detailed lists of configurations and system activities.

Monitor pages support monitoring of infrastructure components—control nodes, virtual routers, analytics nodes, and config nodes. Additionally, users can monitor networking and debug components.

Use the menu options available from the **Monitor** icon to configure and view the statistics you need for better understanding of the activities in your system. See

[Figure 185 on page 566](#)

Figure 185: Monitor Menu



See [Table 58 on page 566](#) for descriptions of the items available under each of the menu options from the **Monitor** icon.

Table 58: Monitor Menu Options

Option	Description
Infrastructure > Dashboard	Shows “at-a-glance” status view of the infrastructure components, including the numbers of virtual routers, control nodes, analytics nodes, and config nodes currently operational, and a bubble chart of virtual routers showing the CPU and memory utilization, log messages, system information, and alerts. See “Monitor > Infrastructure > Dashboard” on page 572 .

Table 58: Monitor Menu Options (*continued*)

Option	Description
Infrastructure > Control Nodes	<p>View a summary for all control nodes in the system, and for each control node, view:</p> <ul style="list-style-type: none"> Graphical reports of memory usage and average CPU load. Console information for a specified time period. A list of all peers with details about type, ASN, and the like. A list of all routes, including next hop, source, local preference, and the like. <p>See "Monitor > Infrastructure > Control Nodes" on page 574.</p>
Infrastructure > Virtual Routers	<p>View a summary of all vRouters in the system, and for each vRouter, view:</p> <ul style="list-style-type: none"> Graphical reports of memory usage and average CPU load. Console information for a specified time period. A list of all interfaces with details such as label, status, associated network, IP address, and the like. A list of all associated networks with their ACLs and VRFs. A list of all active flows with source and destination details, size, and time. <p>See "Monitor > Infrastructure > Virtual Routers" on page 581.</p>
Infrastructure > Analytics Nodes	<p>View activity for the analytics nodes, including memory and CPU usage, analytics host names, IP address, status, and more. See "Monitor > Infrastructure > Analytics Nodes" on page 591.</p>
Infrastructure > Config Nodes	<p>View activity for the config nodes, including memory and CPU usage, config host names, IP address, status, and more. See "Monitor > Infrastructure > Config Nodes" on page 596.</p>
Networking > Networks	<p>For all virtual networks for all projects in the system, view graphical traffic statistics, including:</p> <ul style="list-style-type: none"> Total traffic in and out. Inter VN traffic in and out. The most active ports, peers, and flows for a specified duration. All traffic ingress and egress from connected networks, including their attached policies. <p>See "Monitor > Networking" on page 599.</p>
Networking > Dashboard	<p>For all virtual networks for all projects in the system, view graphical traffic statistics, including:</p> <ul style="list-style-type: none"> Total traffic in and out. Inter VN traffic in and out. <p>You can view the statistics in varying levels of granularity, for example, for a whole project, or for a single network. See "Monitor > Networking" on page 599.</p>
Networking > Projects	<p>View essential information about projects in the system including name, associated networks, and traffic in and out.</p>

Table 58: Monitor Menu Options (*continued*)

Option	Description
Networking > Networks	View essential information about networks in the system including name and traffic in and out.
Networking > Instances	View essential information about instances in the system including name, associated networks, interfaces, vRouters, and traffic in and out.
Debug > Packet Capture	<ul style="list-style-type: none"> • Add and manage packet analyzers. • Attach packet captures and configure their details. • View a list of all packet analyzers in the system and the details of their configurations, including source and destination networks, ports, and IP addresses.

**Related
Documentation**

- [Monitor > Infrastructure > Dashboard on page 572](#)
- [Monitor > Infrastructure > Control Nodes on page 574](#)
- [Monitor > Infrastructure > Virtual Routers on page 581](#)
- [Monitor > Networking on page 599](#)
- [Query > Logs on page 614](#)
- [Query > Flows on page 607](#)

Debugging Processes Using the Contrail Introspect Feature

This topic describes how to use the Sandesh infrastructure and the Contrail Introspect feature to debug processes.

Introspect is a mechanism for taking a program object and querying information about it.

Sandesh is the name of a unified infrastructure in the Contrail Virtual Networking solution.

Sandesh is a way for the Contrail daemons to provide a request-response mechanism. Requests and responses are defined in Sandesh format and the Sandesh compiler generates code to process the requests and send responses.

Sandesh also provides a way to use a Web browser to send Sandesh requests to a Contrail daemon and get the Sandesh responses. This feature is used to debug processes by looking into the operational status of the daemons.

Each Contrail daemon starts an HTTP server, with the following page types:

- The main index.html listing all Sandesh modules and the links to them.
- Sandesh module pages that present HTML forms for each Sandesh request.

- XML-based dynamically-generated pages that display Sandesh responses.
- An automatically generated page that shows all code needed for rendering and all HTTP server-client interactions.

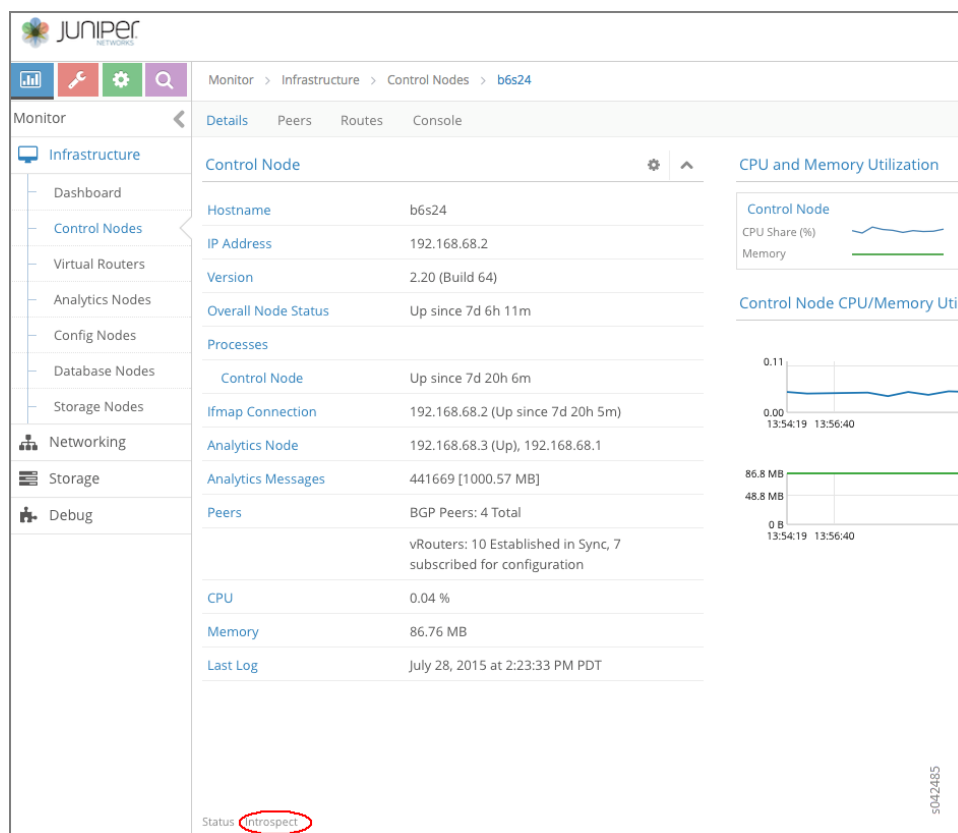
You can display the HTTP introspect of a Contrail daemon directly by accessing the following Introspect ports:

- `<controller-ip>:8083`. This port displays the *contrail-control* introspect port.
- `<compute-ip>:8085`. This port displays the *contrail-vrouter-agent* introspect port.

Another way to launch the Introspect page is by browsing to a particular node page using the Contrail Web user interface.

Figure 186 on page 569 shows the *contrail-control* infrastructure page. Notice the Introspect link at the bottom of the Control Nodes Details tab window.

Figure 186: Control Nodes Details Tab Window



The following are the Sandesh modules for the Contrail control process (*contrail-control*) Introspect port.

- `bgp_peer.xml`
- `control_node.xml`
- `cpuinfo.xml`

- discovery_client_stats.xml
- ifmap_log.xml
- ifmap_server_show.xml
- rtarget_group.xml
- sandesh_trace.xml
- sandesh_uve.xml
- service_chaining.xml
- static_route.xml
- task.xml
- xmpp_server.xml

Figure 187 on page 570 shows the Controller Introspect window.

Figure 187: Controller Introspect Window

The screenshot shows the 'Controller Introspect' window in the Contrail interface. The title bar includes 'Contrail' and navigation buttons: 'Collapse', 'Expand', 'Wrap', and 'NoWrap'. The main content area has a title 'Controller Introspect'. On the left, there is a list of introspectable components, with 'AgentXmppConnectionStatusReq' selected. The main area displays the selected component's name, 'AgentXmppConnectionStatusReq', and a 'Send' button. A small version number 'v0.2.188' is visible in the bottom right corner.

Figure 188 on page 570 shows an example of the BGP Peer (bgp_peer.xml) Introspect page.

Figure 188: BGP Peer Introspect Page

The screenshot shows the 'Bgp_peer Introspect' page in the Contrail interface. The title bar includes 'Contrail' and navigation buttons: 'Collapse', 'Expand', 'Wrap', and 'NoWrap'. The main content area has a title 'Bgp_peer Introspect'. On the left, there is a list of introspectable components, including 'BgpNeighborReq', 'ShowBgpNeighborSummaryReq', 'ClearBgpNeighborReq', 'ShowRouteReq', 'ShowRouteSummaryReq', 'ShowRoutingInstanceReq', 'ShowRoutingInstanceSummaryReq', 'ShowMulticastManagerReq', 'ShowBgpInstanceConfigReq', 'ShowBgpNeighborConfigReq', and 'ShowBgpServerReq'. The main area displays three sections: 'BgpNeighborReq' with input fields for 'neighbor(string)' and 'domain(string)' and a 'Send' button; 'ShowBgpNeighborSummaryReq' with a 'Send' button; and 'ClearBgpNeighborReq' with an input field for 'name(string)' and a 'Send' button. A small version number '9872405' is visible in the bottom right corner.

Figure 189 on page 571 shows an example of the BGP Neighbor Summary Introspect page.

Figure 189: BGP Neighbor Summary Introspect Page

Contrail									
ShowBgpNeighborSummaryResp									
neighbors									
peer	deleted	deleted_at	peer_address	peer_id	peer_asn	encoding	peer_type	state	local_address
b6s23	false	-	192.168.68.1	192.168.68.1	64512	BGP	internal	Established	192.168.68.2
b6s25	false	-	192.168.68.3	192.168.68.3	64512	BGP	internal	Established	192.168.68.2
mx1	false	-	192.168.100.1	192.168.100.1	64512	BGP	internal	Established	192.168.68.2
mx2	false	-	192.168.100.2	192.168.100.2	64512	BGP	internal	Established	192.168.68.2
b6s28	false	-	192.168.68.6	-	0	XMPP	internal	Established	192.168.68.2
b6s18	false	-	192.168.69.5	-	0	XMPP	internal	Established	192.168.68.2
b6s13	false	-	192.168.69.8	-	0	XMPP	internal	Established	192.168.68.2
b6s7	false	-	192.168.69.11	-	0	XMPP	internal	Established	192.168.68.2
b6s33	false	-	192.168.68.11	-	0	XMPP	internal	Established	192.168.68.2
b6s9	false	-	192.168.69.10	-	0	XMPP	internal	Established	192.168.68.2
b6s26	false	-	192.168.68.4	-	0	XMPP	internal	Established	192.168.68.2

The following are the Sandesh modules for the Contrail vRouter agent (**contrail-vrouter-agent**) Introspect port.

- agent.xml
- agent_stats_interval.xml
- cfg.xml
- controller.xml
- cpuinfo.xml
- diag.xml
- discovery_client_stats.xml
- flow_stats_interval.xml
- ifmap_agent.xml
- kstate.xml
- multicast.xml
- pkt.xml
- port_ipc.xml
- sandesh_trace.xml
- sandesh_uve.xml
- services.xml
- stats_interval.xml
- task.xml
- xmpp_server.xml

Figure 190 on page 572 shows an example of the Agent (agent.xml) Introspect page.

Figure 190: Agent Introspect Page

Contrail									
AgentXmppConnectionStatus									
peer									
controller_ip	state	cfg_controller	mcast_controller	last_state	last_event	last_state_at	flap_count	flap_time	rx
192.168.68.3	Established	Yes	No	OpenSent	xmsm::EvXmppKeepAlive	2015-Jul-21 01:20:57.616019	2	2015-Jul-21 01:20:57.555077	rx
192.168.68.2	Established	No	Yes	OpenSent	xmsm::EvXmppKeepAlive	2015-Jul-21 01:20:59.599875	2	2015-Jul-21 01:20:59.548692	rx

Monitor > Infrastructure > Dashboard

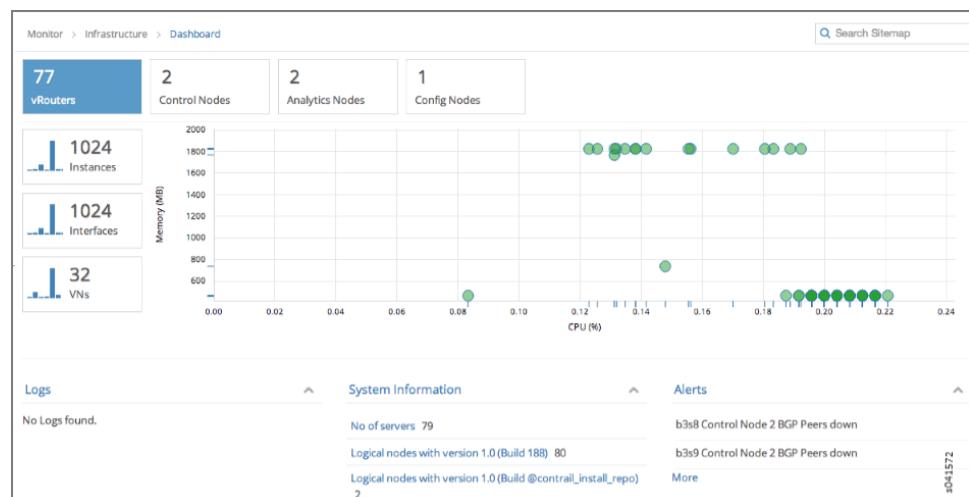
Use **Monitor > Infrastructure > Dashboard** to get an “at-a-glance” view of the system infrastructure components, including the numbers of virtual routers, control nodes, analytics nodes, and config nodes currently operational, a bubble chart of virtual routers showing the CPU and memory utilization, log messages, system information, and alerts.

- [Monitor Dashboard on page 572](#)
- [Monitor Individual Details from the Dashboard on page 573](#)
- [Using Bubble Charts on page 573](#)
- [Color-Coding of Bubble Charts on page 574](#)

Monitor Dashboard

Click **Monitor > Infrastructure > Dashboard** on the left to view the **Dashboard**. See [Figure 191 on page 572](#).

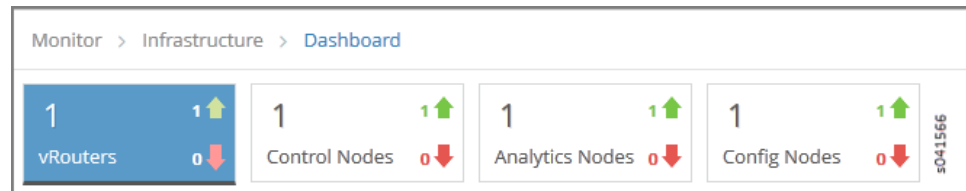
Figure 191: Monitor > Infrastructure > Dashboard



Monitor Individual Details from the Dashboard

Across the top of the **Dashboard** screen are summary boxes representing the components of the system that are shown in the statistics. See [Figure 192 on page 573](#). Any of the control nodes, virtual routers, analytics nodes, and config nodes can be monitored individually and in detail from the **Dashboard** by clicking an associated box, and drilling down for more detail.

Figure 192: Dashboard Summary Boxes



Detailed information about monitoring each of the areas represented by the boxes is provided in the links in [Table 59 on page 573](#).

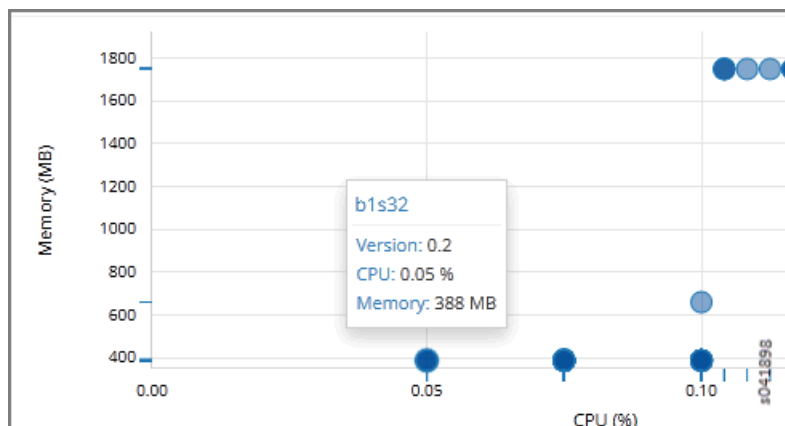
Table 59: Dashboard Summary Boxes

Box	For More Information
vRouters	"Monitor > Infrastructure > Virtual Routers" on page 581
Control Nodes	"Monitor > Infrastructure > Control Nodes" on page 574
Analytics Nodes	"Monitor > Infrastructure > Analytics Nodes" on page 591
Config Nodes	"Monitor > Infrastructure > Config Nodes" on page 596

Using Bubble Charts

Bubble charts show the CPU and memory utilization of components contributing to the current analytics display, including vRouters, control nodes, config nodes, and the like. You can hover over any bubble to get summary information about the component it represents; see [Figure 193 on page 574](#). You can click through the summary information to get more details about the component.

Figure 193: Bubble Summary Information



Color-Coding of Bubble Charts

Bubble charts use the following color-coding scheme:

Control Nodes

- Blue—working as configured.
- Red—error, at least one configured peer is down.

vRouters

- Blue—working, but no instance is launched.
- Green—working with at least one instance launched.
- Red—error, there is a problem with connectivity or a vRouter is in a failed state.

Related Documentation

- [Monitor > Infrastructure > Virtual Routers on page 581](#)
- [Monitor > Infrastructure > Control Nodes on page 574](#)
- [Monitor > Infrastructure > Analytics Nodes on page 591](#)
- [Monitor > Infrastructure > Config Nodes on page 596](#)

Monitor > Infrastructure > Control Nodes

Use **Monitor > Infrastructure > Control Nodes** to gain insight into usage statistics for control nodes.

- [Monitor Control Nodes Summary on page 575](#)
- [Monitor Individual Control Node Details on page 575](#)
- [Monitor Individual Control Node Console on page 577](#)
- [Monitor Individual Control Node Peers on page 579](#)
- [Monitor Individual Control Node Routes on page 580](#)

Monitor Control Nodes Summary

Select **Monitor > Infrastructure > Control Nodes** to see a graphical chart of average memory usage versus average CPU percentage usage for all control nodes in the system. Also on this screen is a list of all control nodes in the system. See [Figure 194 on page 575](#). See [Table 60 on page 575](#) for descriptions of the fields on this screen.

Figure 194: Control Nodes Summary



Table 60: Control Nodes Summary Fields

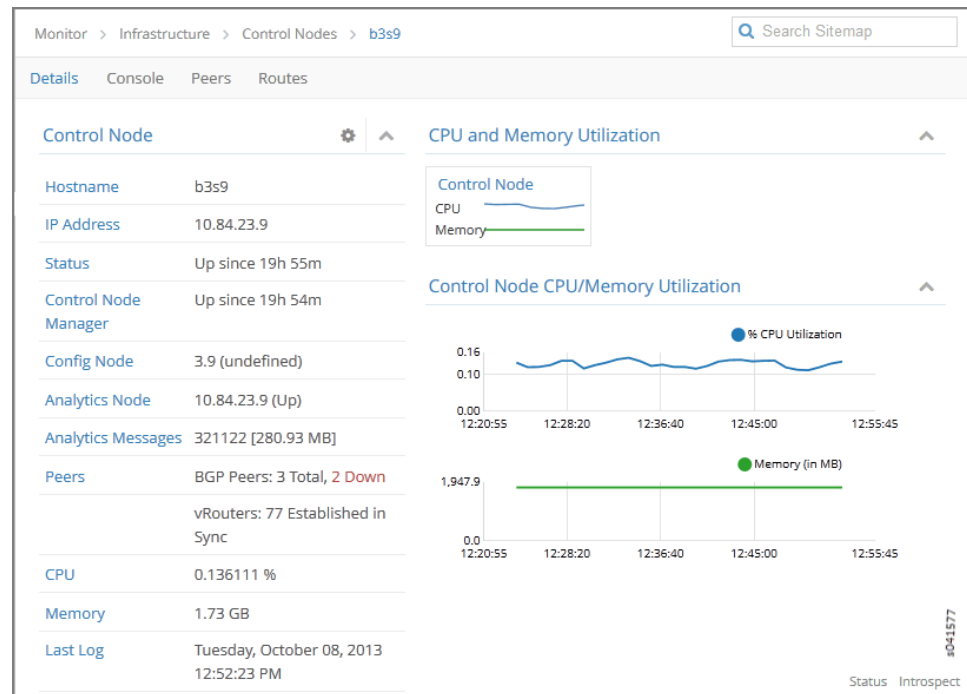
Field	Description
Host name	The name of the control node.
IP Address	The IP address of the control node.
Version	The software version number that is installed on the control node.
Status	The current operational status of the control node — Up or Down.
CPU (%)	The CPU percentage currently in use by the selected control node.
Memory	The memory in MB currently in use and the total memory available for this control node.
Total Peers	The total number of peers for this control node.
Established in Sync Peers	The total number of peers in sync for this control node.
Established in Sync vRouters	The total number of vRouters in sync for this control node.

Monitor Individual Control Node Details

Click the name of any control nodes listed under the **Control Nodes** title to view an array of graphical reports of usage and numerous details about that node. There are several

tabs available to help you probe into more details about the selected control node. The first tab is the **Details** tab; see [Figure 195 on page 576](#).

Figure 195: Individual Control Node—Details Tab



The Details tab provides a summary of the status and activity on the selected node, and presents graphical displays of CPU and memory usage. See [Table 61 on page 576](#) for descriptions of the fields on this tab.

Table 61: Individual Control Node—Details Tab Fields

Field	Description
Hostname	The host name defined for this control node.
IP Address	The IP address of the selected node.
Status	The operational status of the control node.
Control Node Manager	The operational status of the control node manager.
Config Node	The IP address of the configuration node associated with this control node.
Analytics Node	The IP address of the node from which analytics (monitor) information is derived.
Analytics Messages	The total number of analytics messages in and out from this node.
Peers	The total number of peers established for this control node and how many are in sync and of what type.

Table 61: Individual Control Node—Details Tab Fields (*continued*)

Field	Description
CPU	The average percent of CPU load incurred by this control node.
Memory	The average memory usage incurred by this control node.
Last Log	The date and time of the last log message issued about this control node.
Control Node CPU/Memory Utilization	A graphic display x, y chart of the average CPU load and memory usage incurred by this control node over time.

Monitor Individual Control Node Console

Click the **Console** tab for an individual control node to display system logging information for a defined time period, with the last 5 minutes of information as the default display. See [Figure 196 on page 577](#).

Figure 196: Individual Control Node—Console Tab

Monitor > Infrastructure > Control Nodes > b3s9

Search Sitemap

Details Console Peers Routes

Console Logs

Time Range: Custom

From Time: Oct 08, 2013 02:26:33 PM

To Time: Oct 08, 2013 02:31:33 PM

Log Category: All

Log Type: any

Log Level: SYS_DEBUG

Limit: Limit 10 mess

Auto Refresh: ☒

Display Logs Reset

Time	Category	Log Type	Log
2013-10-08 14:31:30:351:353	BGP	BgpStateMachineSessionMessageLog	Bgp Peer 10.84.23.252 : P fsm::EvConnectTimerExp
2013-10-08 14:31:27:971:482	BGP	BgpStateMachineSessionMessageLog	Bgp Peer 10.84.23.253 : P state Connect
2013-10-08 14:31:24:970:157	BGP	BgpStateMachineSessionMessageLog	Bgp Peer 10.84.23.253 : P fsm::EvConnectTimerExp
2013-10-08 14:30:58:220:866	BGP	BgpStateMachineSessionMessageLog	Bgp Peer 10.84.23.252 : P state Connect

See [Table 62 on page 577](#) for descriptions of the fields on the **Console** tab screen.

Table 62: Control Node: Console Tab Fields

Field	Description
Time Range	Select a timeframe for which to review logging information as sent to the console. There are 11 options, ranging from the Last 5 mins through to the Last 24 hrs . The default display is for the Last 5 mins .

Table 62: Control Node: Console Tab Fields (*continued*)

Field	Description
Log Category	Select a log category to display: All _default_ XMPP TCP
Log Type	Select a log type to display.
Log Level	Select a log severity level to display: SYS_EMERG SYS_ALERT SYS_CRIT SYS_ERR SYS_WARN SYS_NOTICE SYS_INFO SYS_DEBUG
Search	Enter any text string to search and display logs containing that string.
Limit	Select from a list an amount to limit the number of messages displayed: No Limit Limit 10 messages Limit 50 messages Limit 100 messages Limit 200 messages Limit 500 messages
Auto Refresh	Click the check box to automatically refresh the display if more messages occur.
Display Logs	Click this button to refresh the display if you change the display criteria.
Reset	Click this button to clear any selected display criteria and reset all criteria to their default settings.
Time	This column lists the time received for each log message displayed.
Category	This column lists the log category for each log message displayed.
Log Type	This column lists the log type for each log message displayed.
Log	This column lists the log message for each log displayed.

Monitor Individual Control Node Peers

The **Peers** tab displays the peers for an individual control node and their peering state. Click the expansion arrow next to the address of any peer to reveal more details. See [Figure 197 on page 579](#).

Figure 197: Individual Control Node—Peers Tab

Peer	Peer Type	Peer ASN	Status	Last flap	Messages (Recv/Sent)
10.84.23.252	BGP	64512	Active, -	-	0/ 0
10.84.23.8	BGP	64512	Established, in sync	-	3754/ 3758
10.84.23.253	BGP	64512	Connect, -	-	0/ 0
10.84.21.4	XMPP	-	Established, in sync	-	2751/ 5189
10.84.21.5	XMPP	-	Established, in sync	-	2753/ 5802
10.84.21.6	XMPP	-	Established, in sync	-	2752/ 4264
10.84.21.34	XMPP	-	Established, in sync	-	2753/ 5659

Details:

```

- {
  name: "b3s9:10.84.21.34",
  value: - {
    XmppPeerInfoData: - {
      state_info: - {
        last_state: "Active",
        state: "Established",
        last_state_at: 1381190447915913
      },
      peer_stats_info: - {

```

See [Table 63 on page 579](#) for descriptions of the fields on the **Peers** tab screen.

Table 63: Control Node: Peers Tab Fields

Field	Description
Peer	The hostname of the peer.
Peer Type	The type of peer.
Peer ASN	The autonomous system number of the peer.
Status	The current status of the peer.
Last flap	The last flap detected for this peer.
Messages (Recv/Sent)	The number of messages sent and received from this peer.

Monitor Individual Control Node Routes

The **Routes** tab displays active routes for this control node and lets you query the results. Use horizontal and vertical scroll bars to view more results. Click the expansion icon next to a routing table name to reveal more details about the selected route. See [Figure 198 on page 580](#).

Figure 198: Individual Control Node—Routes Tab

Routing Table	Prefix	Protocol	Source	Next hop	Label	Secur...	Origin VN
bgp.l3vpn.0	10.84.21.1:13:192.168.30.240/32	XMPP	b1s1	10.84.21.1	28	3	default-domaindemo.v n30
		BGP	10.84.23.9	10.84.21.1	28	3	default-domaindemo.v n30
	10.84.21.1:14:192.168.31.242/32	XMPP	b1s1	10.84.21.1	29	3	default-domaindemo.v n31
		BGP	10.84.23.9	10.84.21.1	29	3	default-domaindemo.v n31
	10.84.21.1:1:192.168.2.231/32	XMPP	b1s1	10.84.21.1	16	3	default-domaindemo.v n2

See [Table 64 on page 580](#) for descriptions of the fields on the **Routes** tab screen.

Table 64: Control Node: Routes Tab Fields

Field	Description
Routing Instance	You can select a single routing instance from a list of all instances for which to display the active routes.
Address Family	Select an address family for which to display the active routes: <ul style="list-style-type: none"> All (default) l3vpn inet inetmcast
(Limit Field)	Select to limit the display of active routes: <ul style="list-style-type: none"> Limit 10 Routes Limit 50 Routes Limit 100 Routes Limit 200 Routes
Peer Source	Select from a list of available peers the peer for which to display the active routes, or select All.

Table 64: Control Node: Routes Tab Fields (*continued*)

Field	Description
Prefix	Enter a route prefix to limit the display of active routes to only those with the designated prefix.
Protocol	Select a protocol for which to display the active routes: All (default) XMPP BGP ServiceChain Static
Display Routes	Click this button to refresh the display of routes after selecting different display criteria.
Reset	Click this button to clear any selected criteria and return the display to default values.
<i>Column</i>	<i>Description</i>
Routing Table	The name of the routing table that stores this route.
Prefix	The route prefix for each active route displayed.
Protocol	The protocol used by the route.
Source	The host source for each active route displayed.
Next hop	The IP address of the next hop for each active route displayed.
Label	The label for each active route displayed.
Security	The security value for each active route displayed.
Origin VN	The virtual network from which the route originates.
AS Path	The AS path for each active route displayed.

Monitor > Infrastructure > Virtual Routers

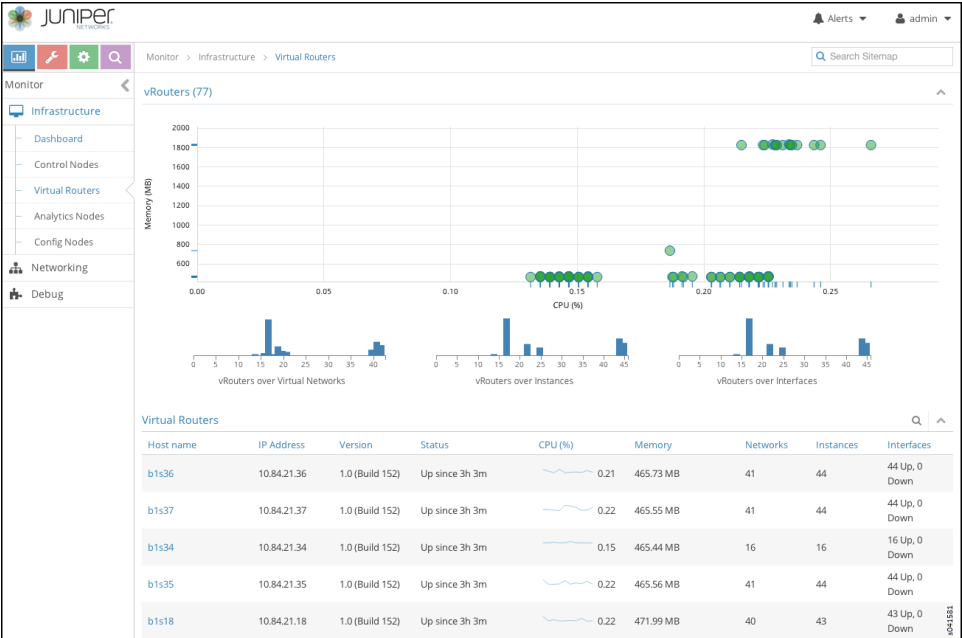
- [Monitor vRouters Summary on page 582](#)
- [Monitor Individual vRouters Tabs on page 583](#)
- [Monitor Individual vRouter Details Tab on page 583](#)
- [Monitor Individual vRouters Interfaces Tab on page 584](#)
- [Monitor Individual vRouters Networks Tab on page 585](#)
- [Monitor Individual vRouters ACL Tab on page 586](#)
- [Monitor Individual vRouters Flows Tab on page 587](#)

- [Monitor Individual vRouters Routes Tab on page 588](#)
- [Monitor Individual vRouter Console Tab on page 589](#)

Monitor vRouters Summary

Click **Monitor > Infrastructure > Virtual Routers** to view the **vRouters** summary screen. See [Figure 199 on page 582](#).

Figure 199: vRouters Summary



See [Table 65 on page 582](#) for descriptions of the fields on the **vRouters Summary** screen.

Table 65: vRouters Summary Fields

Field	Description
Host name	The name of the vRouter. Click the name of any vRouter to reveal more details.
IP Address	The IP address of the vRouter.
Version	The version of software installed on the system.
Status	The current operational status of the vRouter — Up or Down.
CPU (%)	The CPU percentage currently in use by the selected vRouter.
Memory (MB)	The memory currently in use and the total memory available for this vRouter.
Networks	The total number of networks for this vRouter.
Instances	The total number of instances for this vRouter.

Table 65: vRouters Summary Fields (*continued*)

Field	Description
Interfaces	The total number of interfaces for this vRouter.

Monitor Individual vRouters Tabs

Click the name of any vRouter to view details about performance and activities for that vRouter. Each individual vRouters screen has the following tabs.

- **Details**—similar display of information as on individual control nodes **Details** tab. See [Figure 200 on page 583](#).
- **Console**—similar display of information as on individual control nodes **Console** tab. See [Figure 206 on page 590](#).
- **Interfaces**—details about associated interfaces. See [Figure 201 on page 585](#).
- **Networks**—details about associated networks. See [Figure 202 on page 586](#).
- **ACL**—details about access control lists. See [Figure 203 on page 587](#).
- **Flows**—details about associated traffic flows. See [Figure 204 on page 588](#).
- **Routes**—details about associated routes. See [Figure 205 on page 589](#).

Monitor Individual vRouter Details Tab

The **Details** tab provides a summary of the status and activity on the selected node, and presents graphical displays of CPU and memory usage; see [Figure 200 on page 583](#). See [Table 66 on page 583](#) for descriptions of the fields on this tab.

Figure 200: Individual vRouters—Details Tab

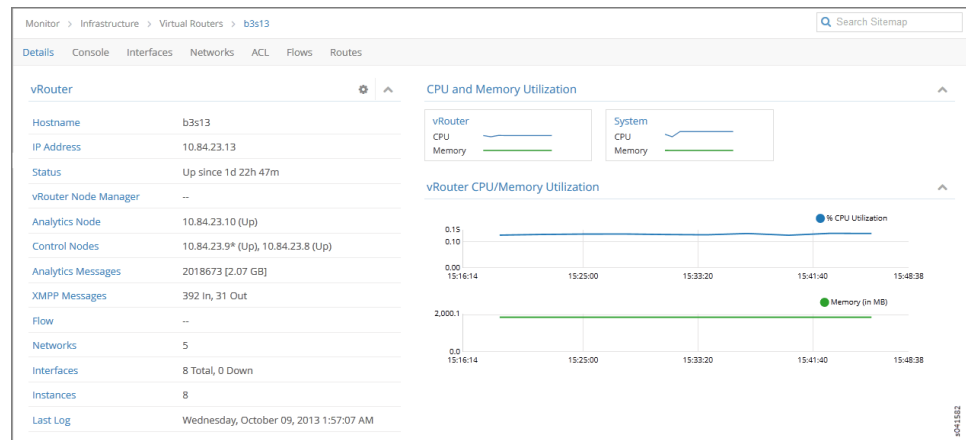


Table 66: vRouters Details Tab Fields

Field	Description
Hostname	The hostname of the vRouter.

Table 66: vRouters Details Tab Fields (*continued*)

Field	Description
IP Address	The IP address of the selected vRouter.
Status	The operational status of the vRouter.
vRouter Node Manager	The operational status of the vRouter node manager.
Analytics Node	The IP address of the node from which analytics (monitor) information is derived.
Control Nodes	The IP address of the configuration node associated with this vRouter.
Analytics Messages	The total number of analytics messages in and out from this node.
XMPP Messages	The total number of XMPP messages that have gone in and out of this vRouter.
Flow	The number of active flows and the total flows for this vRouter.
Networks	The number of networks associated with this vRouter.
Interfaces	The number of interfaces associated with this vRouter.
Instances	The number of instances associated with this vRouter.
Last Log	The date and time of the last log message issued about this vRouter.
vRouter CPU/Memory Utilization	Graphs (x, y) displaying CPU and memory utilization averages over time for this vRouter, in comparison to system utilization averages.

Monitor Individual vRouters Interfaces Tab

The **Interfaces** tab displays details about the interfaces associated with an individual vRouter. Click the expansion arrow next to any interface name to reveal more details. Use horizontal and vertical scroll bars to access all portions of the screen. See [Figure 201 on page 585](#). See [Table 67 on page 585](#) for descriptions of the fields on the **Interfaces** tab screen.

Figure 201: Individual vRouters—Interfaces Tab

Monitor > Infrastructure > Virtual Routers > b1s36

Search Sitemap

Details Console **Interfaces** Networks ACL Flows Routes

Interfaces

Name	Label	Status	Network	IP Address	Floating IP	Instance
tap25e5cee3-07	18	Up	default-domain:demo:vn30	192.168.30.247	None	005132fd-0d83-4db7-88c8-bd49d68e9480
tap4d91aab1-f1	25	Up	default-domain:demo:vn26	192.168.26.247	None	65d6c6e9-7a82-43d8-a706-f74d81715920
tap5a8cd9dd-5b	27	Up	default-domain:demo:vn23	192.168.23.249	None	a159c518-4fb6-402a-ae0d-eb5b4457b551
tap603a5e0b-8b	16	Up	default-domain:demo:vn19	192.168.19.247	None	fe622580-b0cf-4c6d-89e5-d2065e7e87e4
tap68ad232c-76	19	Up	default-domain:demo:vn28	192.168.28.247	None	91089d89-76b5-46c2-abc9-b9693bcb37ac

Details :

```
{
  index: "6",
  name: "tap68ad232c-76",
  uuid: "68ad232c-76d1-4fe2-a200-42182497545e",
  vrf_name: "default-domain:demo:vn28:vn28",
  active: "Active",
  dhcp_service: "Enable",
}
```

Table 67: vRouters: Interfaces Tab Fields

Field	Description
Name	The name of the interface.
Label	The label for the interface.
Status	The current status of the interface.
Network	The network associated with the interface.
IP Address	The IP address of the interface.
Floating IP	Displays any floating IP addresses associated with the interface.
Instance	The name of any instance associated with the interface.

Monitor Individual vRouters Networks Tab

The **Networks** tab displays details about the networks associated with an individual vRouter. Click the expansion arrow at the name of any network to reveal more details. See [Figure 202 on page 586](#). See [Table 68 on page 586](#) for descriptions of the fields on the **Networks** tab screen.

Figure 202: Individual vRouters—Networks Tab

Name	ACLs	VRF
default-domain:demo:vn24	a372751f-6497-41e9-b409-fa4ab5ce6b7f	default-domain:demo:vn24:vn24
default-domain:demo:vn22	195af177-0a28-49a1-9cf0-2ceac22af5a1	default-domain:demo:vn22:vn22
default-domain:demo:vn30	362cce6e-2894-42d6-ba03-3ee98cac8809	default-domain:demo:vn30:vn30
default-domain:demo:vn21	5918a068-1cd5-4993-9cff-386a807940ca	default-domain:demo:vn21:vn21
default-domain:demo:vn28	dd87c461-97c0-4d47-bff0-89040e7d6ab0	default-domain:demo:vn28:vn28
default-domain:demo:vn19	f0465432-6fc0-4fb3-967c-392100617408	default-domain:demo:vn19:vn19
default-domain:demo:vn2	1c46e7e0-f799-4bc6-ae09-e4654c263aa6	default-domain:demo:vn2:vn2

```

- {
  name: "default-domain:demo:vn2",
  uuid: "63d08f7a-b342-4892-9171-edab9f4c397f",
  acl_uuid: "1c46e7e0-f799-4bc6-ae09-e4654c263aa6",
  mirror_acl_uuid: - {},
  mirror_cfg_acl_uuid: - {},
  vrf_name: "default-domain:demo:vn2:vn2",
  ipam_data: - {
    list: - {

```

Table 68: vRouters: Networks Tab Fields

Field	Description
Name	The name of each network associated with this vRouter.
ACLs	The name of the access control list associated with the listed network.
VRF	The identifier of the VRF associated with the listed network.
Action	Click the icon to select the action: Edit, Delete

Monitor Individual vRouters ACL Tab

The **ACL** tab displays details about the access control lists (ACLs) associated with an individual vRouter. Click the expansion arrow next to the UUID of any ACL to reveal more details. See [Figure 203 on page 587](#). See [Table 69 on page 587](#) for descriptions of the fields on the **ACL** tab screen.

Figure 203: Individual vRouters—ACL Tab

Monitor > Infrastructure > Virtual Routers > b1s36

Search Sitemap

Details Console Interfaces Networks **ACL** Flows Routes

ACL

UUID	Flows	Action	Protocol	Source Network or Prefix	Source Port	Destination Network or Prefix	D
195af177-0a28-49a1-9cf0-2ce-ac22af5a1	8	pass	any	-	any	-	a
		pass	any	-	any	-	a
		pass	any	-	any	-	a
1c46e7e0-f799-4bc6-ae09-e4654c263aa6	8	pass	any	-	any	-	a

Details :

```

- {
  uuid: "1c46e7e0-f799-4bc6-ae09-e4654c263aa6",
  dynamic_acl: "false",
  entries: - {
    list: - {
      ACLEntrySandeshData: - [
        - {
          ace_id: "1",

```

Table 69: vRouters: ACL Tab Fields

Field	Description
UUID	The universal unique identifier (UUID) associated with the listed ACL.
Flows	The flows associated with the listed ACL.
Action	The traffic action defined by the listed ACL.
Protocol	The protocol associated with the listed ACL.
Source Network or Prefix	The name or prefix of the source network associated with the listed ACL.
Source Port	The source port associated with the listed ACL.
Destination Network or Prefix	The name or prefix of the destination network associated with the listed ACL.
Destination Port	The destination port associated with the listed ACL.
ACE Id	The ACE ID associated with the listed ACL.

Monitor Individual vRouters Flows Tab

The **Flows** tab displays details about the flows associated with an individual vRouter. Click the expansion arrow next to any ACL/SG UUID to reveal more details. Use the horizontal and vertical scroll bars to access all portions of the screen. See

Figure 204 on page 588. See Table 70 on page 588 for descriptions of the fields on the **Flows** tab screen.

Figure 204: Individual vRouters—Flows Tab

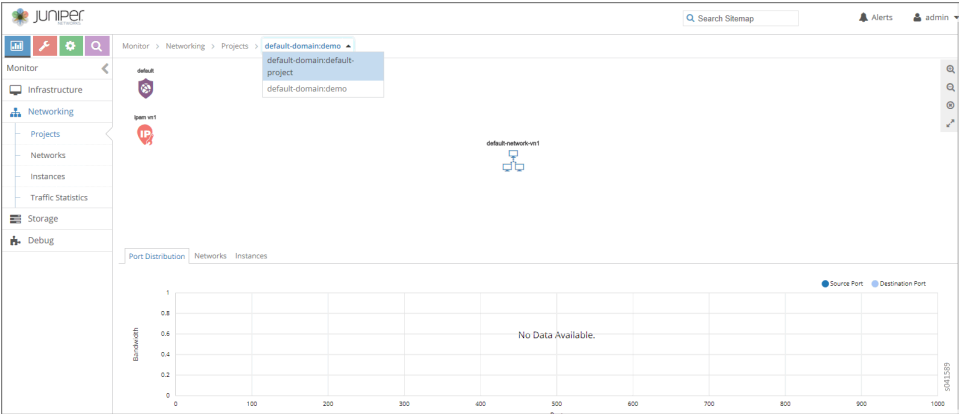


Table 70: vRouters: Flows Tab Fields

Field	Description
ACL UUID	The default is to show All flows, however, you can select from a drop down list any single flow to view its details.
ACL / SG UUID	The universal unique identifier (UUID) associated with the listed ACL or SG.
Protocol	The protocol associated with the listed flow.
Src Network	The name of the source network associated with the listed flow.
Src IP	The source IP address associated with the listed flow.
Src Port	The source port of the listed flow.
Dest Network	The name of the destination network associated with the listed flow.
Dest IP	The destination IP address associated with the listed flow.
Dest Port	The destination port associated with the listed flow.
Bytes/Pkts	The number of bytes and packets associated with the listed flow.
Setup Time	The setup time associated with the listed flow.

Monitor Individual vRouters Routes Tab

The **Routes** tab displays details about unicast and multicast routes in specific VRFs for an individual vRouter. Click the expansion arrow next to the route prefix to reveal more details. See Figure 205 on page 589. See Table 71 on page 589 for descriptions of the fields on the **Routes** tab screen.

Figure 205: Individual vRouters—Routes Tab

Monitor > Infrastructure > Virtual Routers > b1s36

Details Console Interfaces Networks ACL Flows Routes

VRF: default-domain:default-project:ip-fabric:default

Show Routes: Unicast Multicast

Prefix	Next hop ...	Next hop details
0.0.0.0 / 0	arp	Interface: p2p0p0 Mac: 40:b4:f0:68:20:4e IP: 10.84.21.254
10.84.21.0 / 24	resolve	Source: Local Destination VN: default-domain:default-project:ip-fabric
10.84.21.1 / 32	arp	Interface: p2p0p0 Mac: 0:25:90:ab:b0:2c IP: 10.84.21.1
10.84.21.2 / 32	arp	Interface: p2p0p0 Mac: 0:25:90:ab:b0:38 IP: 10.84.21.2
10.84.21.3 / 32	arp	Interface: p2p0p0 Mac: 0:25:90:ab:af:ce IP: 10.84.21.3
10.84.21.4 / 32	arp	Interface: p2p0p0 Mac: 0:25:90:ab:ae:82 IP: 10.84.21.4
10.84.21.5 / 32	arp	Interface: p2p0p0 Mac: 0:25:90:ab:b0:16 IP: 10.84.21.5

Details:

```
{
  "displayPrefix": "10.84.21.5 / 32",
  "path": {
    "nh": {
      "type": "arp",
      "ref_count": "1",
      "valid": "true",
      "policy": "disabled",
      "sip": "10.84.21.5",
      "vrf": "default-domain:default-project:ip-fabric:default"
    }
  }
}
```

Table 71: vRouters: Routes Tab Fields

Field	Description
VRF	Select from a drop down list the virtual routing and forwarding (VRF) to view.
Show Routes	Select to show the route type: Unicast or Multicast .
Prefix	The IP address prefix of a route.
Next hop	The next hop method for this route.
Next hop details	The next hop details for this route.

Monitor Individual vRouter Console Tab

Click the **Console** tab for an individual vRouter to display system logging information for a defined time period, with the last 5 minutes of information as the default display. See [Figure 206 on page 590](#). See [Table 72 on page 590](#) for descriptions of the fields on the **Console** tab screen.

Figure 206: Individual vRouter—Console Tab

Monitor > Infrastructure > Virtual Routers > b1s36

Details Console Interfaces Networks ACL Flows Routes

Console Logs

Time Range: Custom From Time: Oct 02, 2013 05:00:39 AM To Time: Oct 02, 2013 05:05:39 AM

Log Category: All Log Type: any Log Level: SYS_INFO Limit: Limit 10 messages Auto Refresh: ☒

Display Logs Reset

Time	Category	Log Type	Log
2013-10-02 05:05:39:572:199	Agent	AgentRouteLog	Added route 192.168.31.222/32 in VRF default-domaindemo:vn31:vn31 10.84.23.9
2013-10-02 05:05:34:761:107	Agent	AgentRouteLog	Added route 192.168.31.224/32 in VRF default-domaindemo:vn31:vn31 10.84.23.9
2013-10-02 05:05:34:731:318	Agent	AgentRouteLog	Added route 192.168.31.223/32 in VRF default-domaindemo:vn31:vn31 10.84.23.9
2013-10-02 05:05:32:283:326	Agent	AgentRouteLog	Added route 192.168.31.225/32 in VRF default-domaindemo:vn31:vn31 10.84.23.8
2013-10-02 05:05:31:282:424	Agent	AgentRouteLog	Added route 192.168.31.227/32 in VRF default-domaindemo:vn31:vn31 10.84.23.8
2013-10-02 05:05:29:319:521	Agent	AgentRouteLog	Added route 192.168.31.229/32 in VRF default-domaindemo:vn31:vn31 10.84.23.9

Table 72: Control Node: Console Tab Fields

Field	Description
Time Range	Select a timeframe for which to review logging information as sent to the console. There are several options, ranging from Last 5 mins through to the Last 24 hrs , plus a Custom time range.
From Time	If you select Custom in Time Range , enter the start time.
To Time	If you select Custom in Time Range , enter the end time.
Log Category	Select a log category to display: <ul style="list-style-type: none"> • All • _default_ • XMPP • TCP
Log Type	Select a log type to display.
Log Level	Select a log severity level to display: <ul style="list-style-type: none"> • SYS_EMERG • SYS_ALERT • SYS_CRIT • SYS_ERR • SYS_WARN • SYS_NOTICE • SYS_INFO • SYS_DEBUG

Table 72: Control Node: Console Tab Fields (*continued*)

Field	Description
Limit	Select from a list an amount to limit the number of messages displayed: <ul style="list-style-type: none"> • No Limit • Limit 10 messages • Limit 50 messages • Limit 100 messages • Limit 200 messages • Limit 500 messages
Auto Refresh	Click the check box to automatically refresh the display if more messages occur.
Display Logs	Click this button to refresh the display if you change the display criteria.
Reset	Click this button to clear any selected display criteria and reset all criteria to their default settings.
<i>Columns</i>	
Time	This column lists the time received for each log message displayed.
Category	This column lists the log category for each log message displayed.
Log Type	This column lists the log type for each log message displayed.
Log	This column lists the log message for each log displayed.

Monitor > Infrastructure > Analytics Nodes

Select **Monitor > Infrastructure > Analytics Nodes** to view the console logs, generators, and query expansion (QE) queries of the analytics nodes.

- [Monitor Analytics Nodes on page 591](#)
- [Monitor Analytics Individual Node Details Tab on page 592](#)
- [Monitor Analytics Individual Node Generators Tab on page 593](#)
- [Monitor Analytics Individual Node QE Queries Tab on page 594](#)
- [Monitor Analytics Individual Node Console Tab on page 595](#)

Monitor Analytics Nodes

Select **Monitor > Infrastructure > Analytics Nodes** to view a summary of activities for the analytics nodes; see [Figure 207 on page 592](#). See [Table 73 on page 592](#) for descriptions of the fields on the analytics summary.

Figure 207: Analytics Nodes Summary



Table 73: Fields on Analytics Nodes Summary

Field	Description
Host name	The name of this node.
IP address	The IP address of this node.
Version	The version of software installed on the system.
Status	The current operational status of the node — Up or Down — and the length of time it is in that state.
CPU (%)	The average CPU percentage usage for this node.
Memory	The average memory usage for this node.
Generators	The total number of generators for this node.

Monitor Analytics Individual Node Details Tab

Click the name of any analytics node displayed on the analytics summary to view the **Details** tab for that node. See [Figure 208 on page 593](#).

See [Table 74 on page 593](#) for descriptions of the fields on this screen.

Figure 208: Monitor Analytics Individual Node Details Tab

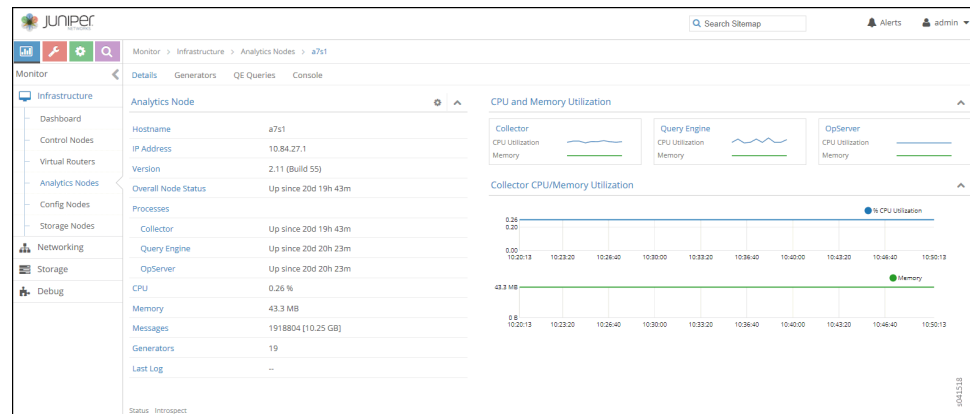


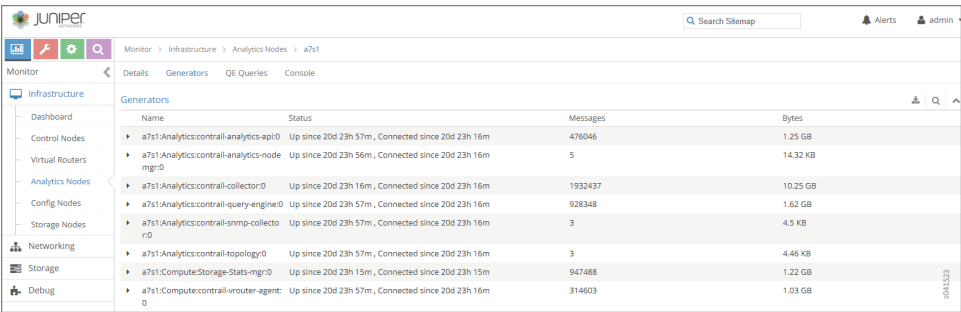
Table 74: Monitor Analytics Individual Node Details Tab Fields

Field	Description
Hostname	The name of this node.
IP Address	The IP address of this node.
Version	The installed version of the software.
Overall Node Status	The current operational status of the node — Up or Down — and the length of time in this state.
Processes	The current status of each analytics process, including Collector, Query Engine, and OpServer.
CPU (%)	The average CPU percentage usage for this node.
Memory	The average memory usage of this node.
Messages	The total number of messages for this node.
Generators	The total number of generators associated with this node.
Last Log	The date and time of the last log message issued about this node.

Monitor Analytics Individual Node Generators Tab

The **Generators** tab displays information about the generators for an individual analytics node; see [Figure 209 on page 594](#). Click the expansion arrow next to any generator name to reveal more details. See [Table 75 on page 594](#) for descriptions of the fields on the **Peers** tab screen.

Figure 209: Individual Analytics Node—Generators Tab



Name	Status	Messages	Bytes
a7s1:Analytics:contrail-analytics-api0	Up since 20d 23h 57m, Connected since 20d 23h 16m	476046	1.25 GB
a7s1:Analytics:contrail-analytics-node-mgr0	Up since 20d 23h 56m, Connected since 20d 23h 16m	5	14.32 KB
a7s1:Analytics:contrail-collector0	Up since 20d 23h 16m, Connected since 20d 23h 16m	1932437	10.25 GB
a7s1:Analytics:contrail-query-engine0	Up since 20d 23h 57m, Connected since 20d 23h 16m	928348	1.62 GB
a7s1:Analytics:contrail-snmp-collector0	Up since 20d 23h 57m, Connected since 20d 23h 16m	3	4.5 KB
a7s1:Analytics:contrail-topology0	Up since 20d 23h 57m, Connected since 20d 23h 16m	3	4.46 KB
a7s1:Compute:Storage-Stats-mgr0	Up since 20d 23h 15m, Connected since 20d 23h 15m	947488	1.22 GB
a7s1:Compute:contrail-vrouter-agent0	Up since 20d 23h 57m, Connected since 20d 23h 16m	314603	1.03 GB

Table 75: Monitor Analytics Individual Node Generators Tab Fields

Field	Description
Name	The host name of the generator.
Status	The current status of the peer— Up or Down — and the length of time in that state.
Messages	The number of messages sent and received from this peer.
Bytes	The total message size in bytes.

Monitor Analytics Individual Node QE Queries Tab

The **QE Queries** tab displays the number of query expansion (QE) messages that are in the queue for this analytics node. See [Figure 210 on page 594](#).

See [Table 76 on page 594](#) for descriptions of the fields on the **QE Queries** tab screen.

Figure 210: Individual Analytics Node—QE QueriesTab



Enqueue Time	Query	Progress
No QE Queries to display		

Table 76: Analytics Node QE Queries Tab Fields

Field	Description
Enqueue Time	The length of time this message has been in the queue waiting to be delivered.
Query	The query message.
Progress (%)	The percentage progress for the message delivery.

Monitor Analytics Individual Node Console Tab

Click the **Console** tab for an individual analytics node to display system logging information for a defined time period. See [Figure 211 on page 595](#). See [Table 77 on page 595](#) for descriptions of the fields on the **Console** tab screen.

Figure 211: Analytics Individual Node—Console Tab

Table 77: Monitor Analytics Individual Node Console Tab Fields

Field	Description
Time Range	Select a timeframe for which to review logging information as sent to the console. There are 11 options, ranging from the Last 5 mins through to the Last 24 hrs . The default display is for the Last 5 mins .
Log Category	Select a log category to display: <ul style="list-style-type: none"> All _default_ XMPP TCP
Log Type	Select a log type to display.
Log Level	Select a log severity level to display: <ul style="list-style-type: none"> SYS_EMERG SYS_ALERT SYS_CRIT SYS_ERR SYS_WARN SYS_NOTICE SYS_INFO SYS_DEBUG
Keywords	Enter any text string to search for and display logs containing that string.

Table 77: Monitor Analytics Individual Node Console Tab Fields (*continued*)

Field	Description
(Limit field)	Select the number of messages to display: No Limit Limit 10 messages Limit 50 messages Limit 100 messages Limit 200 messages Limit 500 messages
Auto Refresh	Click the check box to automatically refresh the display if more messages occur.
Display Logs	Click this button to refresh the display if you change the display criteria.
Reset	Click this button to clear any selected display criteria and reset all criteria to their default settings.
Time	This column lists the time received for each log message displayed.
Category	This column lists the log category for each log message displayed.
Log Type	This column lists the log type for each log message displayed.
Log	This column lists the log message for each log displayed.

Monitor > Infrastructure > Config Nodes

Select **Monitor > Infrastructure > Config Nodes** to view the information about the system config nodes.

- [Monitor Config Nodes on page 596](#)
- [Monitor Individual Config Node Details on page 597](#)
- [Monitor Individual Config Node Console on page 598](#)

Monitor Config Nodes

Select **Monitor > Infrastructure > Config Nodes** to view a summary of activities for the analytics nodes. See [Figure 212 on page 597](#).

Figure 212: Config Nodes Summary

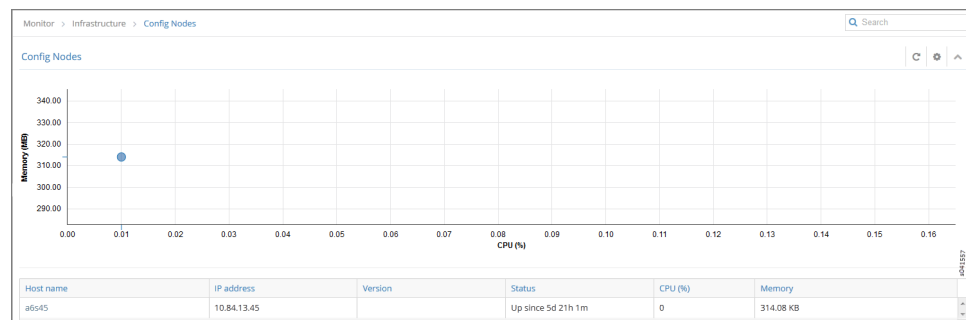


Table 78 on page 597 describes the fields in the Config Nodes summary.

Table 78: Config Nodes Summary Fields

Field	Description
Host name	The name of this node.
IP address	The IP address of this node.
Version	The version of software installed on the system.
Status	The current operational status of the node — Up or Down — and the length of time it is in that state.
CPU (%)	The average CPU percentage usage for this node.
Memory	The average memory usage for this node.

Monitor Individual Config Node Details

Click the name of any config node displayed on the config nodes summary to view the **Details** tab for that node; see [Figure 213 on page 597](#).

Figure 213: Individual Config Nodes— Details Tab

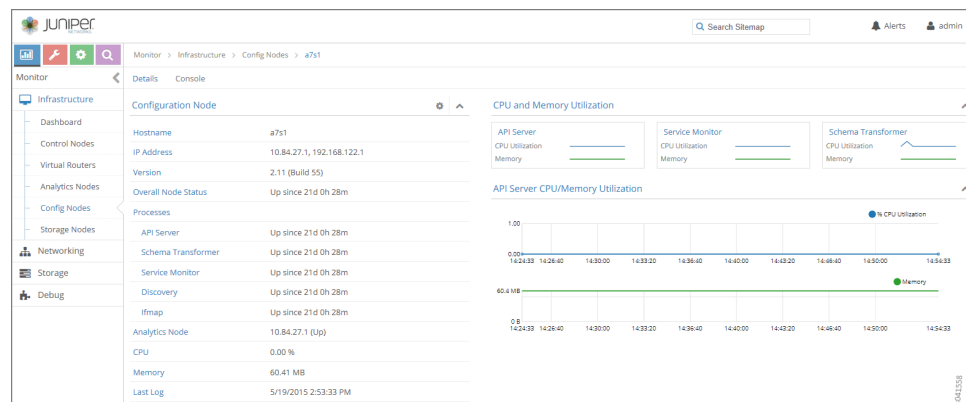


Table 79 on page 598 describes the fields on the Details screen.

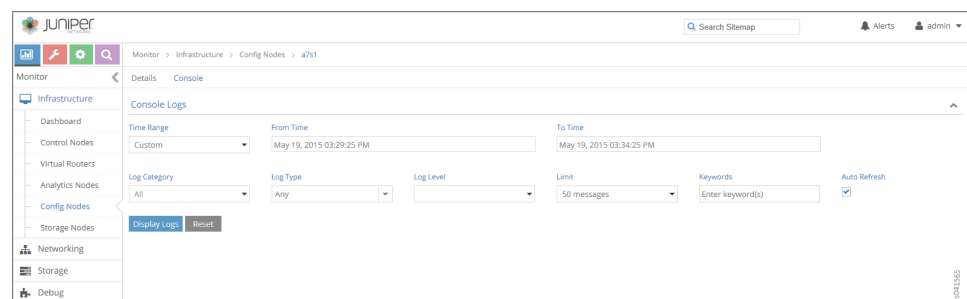
Table 79: Individual Config Nodes— Details Tab Fields

Field	Description
Hostname	The name of the config node.
IP Address	The IP address of this node.
Version	The installed version of the software.
Overall Node Status	The current operational status of the node — Up or Down — and the length of time it is in this state.
Processes	The current operational status of the processes associated with the config node, including AI Server, Schema Transformer, Service Monitor, and the like.
Analytics Node	The analytics node associated with this node.
CPU (%)	The average CPU percentage usage for this node.
Memory	The average memory usage by this node.

Monitor Individual Config Node Console

Click the **Console** tab for an individual config node to display system logging information for a defined time period. See [Figure 214 on page 598](#).

Figure 214: Individual Config Node—Console Tab



See [Table 80 on page 598](#) for descriptions of the fields on the **Console** tab screen.

Table 80: Individual Config Node-Console Tab Fields

Field	Description
Time Range	Select a timeframe for which to review logging information as sent to the console. Use the drop down calendar in the fields From Time and To Time to select the date and times to include in the time range for viewing.
Log Category	Select from the drop down menu a log category to display. The option to view All is also available.
Log Type	Select a log type to display.

Table 80: Individual Config Node-Console Tab Fields (*continued*)

Field	Description
Log Level	Select a log severity level to display:
Limit	Select from a list an amount to limit the number of messages displayed: All Limit 10 messages Limit 50 messages Limit 100 messages Limit 200 messages Limit 500 messages
Keywords	Enter any key words by which to filter the log messages displayed.
Auto Refresh	Click the check box to automatically refresh the display if more messages occur.
Display Logs	Click this button to refresh the display if you change the display criteria.
Reset	Click this button to clear any selected display criteria and reset all criteria to their default settings.

Monitor > Networking

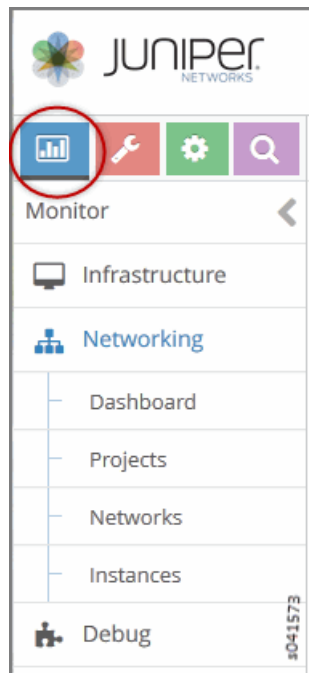
The **Monitor -> Networking** pages give an overview of the networking traffic statistics and health of domains, projects within domains, virtual networks within projects, and virtual machines within virtual networks.

- [Monitor > Networking Menu Options on page 599](#)
- [Monitor -> Networking -> Dashboard on page 600](#)
- [Monitor > Networking > Projects on page 601](#)
- [Monitor Projects Detail on page 602](#)
- [Monitor > Networking > Networks on page 604](#)

Monitor > Networking Menu Options

[Figure 215 on page 600](#) shows the menu options available under **Monitor > Networking**.

Figure 215: Monitor Networking Menu Options



Monitor -> Networking -> Dashboard

Select **Monitor -> Networking -> Dashboard** to gain insight into usage statistics for domains, virtual networks, projects, and virtual machines. When you select this option, the Traffic Statistics for Domain window is displayed as shown in [Figure 216 on page 600](#).

Figure 216: Traffic Statistics for Domain Window

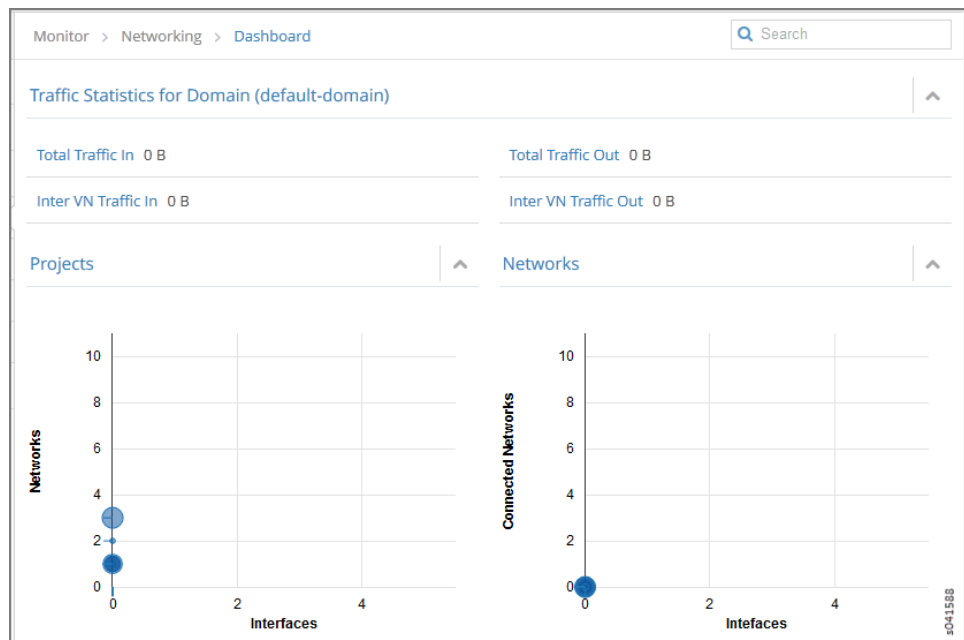


Table 81 on page 601 describes the fields in the Traffic Statistics for Domain window.

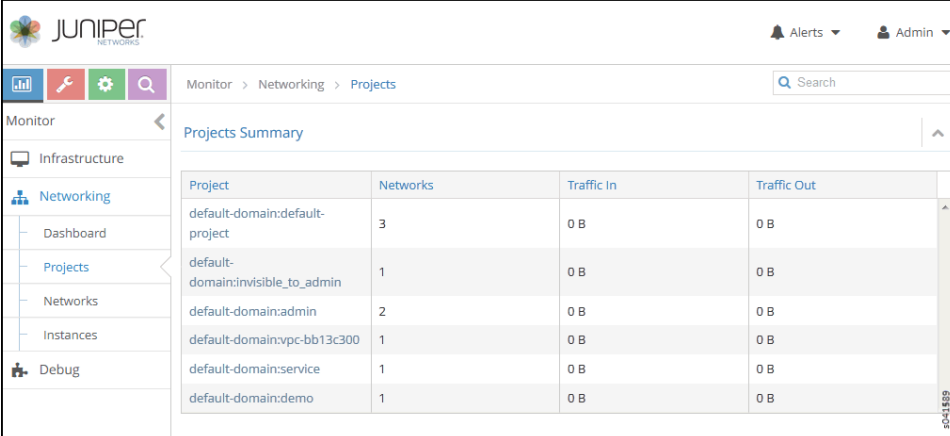
Table 81: Projects Summary Fields

Field	Description
Total Traffic In	The volume of traffic into this domain
Total Traffic Out	The volume of traffic out of this domain.
Inter VN Traffic In	The volume of inter-virtual network traffic into this domain.
Inter VN Traffic Out	The volume of inter-virtual network traffic out of this domain.
Projects	This chart displays the networks and interfaces for projects with the most throughput over the past 30 minutes. Click Projects then select Monitor > Networking > Projects , to display more detailed statistics.
Networks	This chart displays the networks for projects with the most throughput over the past 30 minutes. Click Networks then select Monitor > Networking > Networks , to display more detailed statistics.

Monitor > Networking > Projects

Select **Monitor > Networking > Projects** to see information about projects in the system. See Figure 217 on page 601.

Figure 217: Monitor > Networking > Projects



Project	Networks	Traffic In	Traffic Out
default-domain:default-project	3	0 B	0 B
default-domain:invisible_to_admin	1	0 B	0 B
default-domain:admin	2	0 B	0 B
default-domain:vpc-bb13c300	1	0 B	0 B
default-domain:service	1	0 B	0 B
default-domain:demo	1	0 B	0 B

See Table 82 on page 601 for descriptions of the fields on this screen.

Table 82: Projects Summary Fields

Field	Description
Projects	The name of the project. You can click the name to access details about connectivity for this project.
Networks	The volume of inter-virtual network traffic out of this domain.

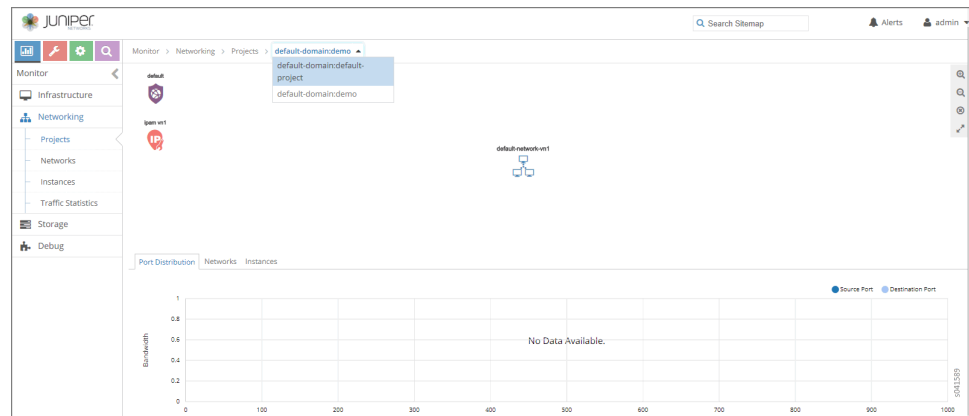
Table 82: Projects Summary Fields (*continued*)

Field	Description
Traffic In	The volume of traffic into this domain.
Traffic Out	The volume of traffic out of this domain.

Monitor Projects Detail

You can click any of the projects listed on the Projects Summary to get details about connectivity, source and destination port distribution, and instances. When you click an individual project, the Summary tab for Connectivity Details is displayed as shown in [Figure 218 on page 602](#). Hover over any of the connections to get more details.

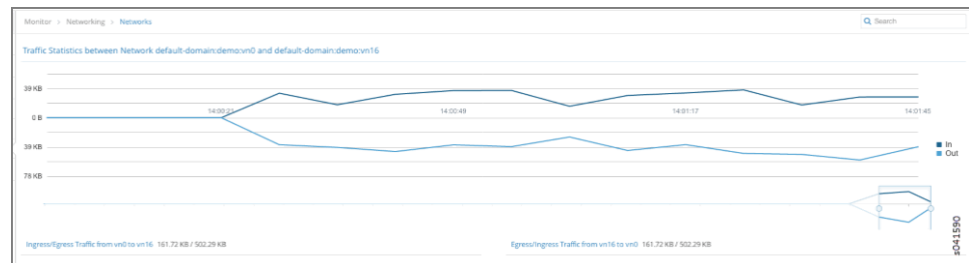
Figure 218: Monitor Projects Connectivity Details



In the Connectivity Details window you can click the links between the virtual networks to view the traffic statistics between the virtual networks.

The Traffic Statistics information is also available when you select **Monitor > Networking > Networks** as shown in [Figure 219 on page 602](#).

Figure 219: Traffic Statistics Between Networks



In the Connectivity Details window you can click the Instances tab to get a summary of details for each of the instances in this project.

Figure 220: Projects Instances Summary

Instance	Virtual Network	Interfaces	vRouter	IP Address	Floating IP	Traffic (In/Out)
out	default-domain:admin:right	1	hp1	2.2.2.252		129.87 KB / 119.83 KB
NAT1_1	default-domain:admin:right	1	hp1	2.2.2.253 250.250.1.253 (1 more)		3.69 MB / 1.15 MB
in	default-domain:admin:left	1	hp1	1.1.1.252		132.75 KB / 122.02 KB

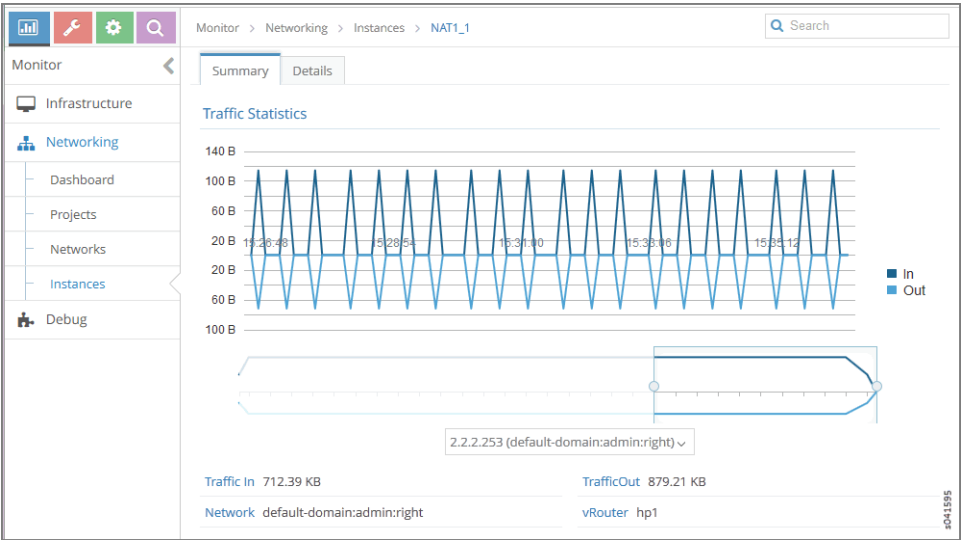
See Table 3 for a description of the fields on this screen.

Table 83: Projects Instances Summary Fields

Field	Description
Instance	The name of the instance. Click the name then select Monitor > Networking > Instances to display details about the traffic statistics for this instance.
Virtual Network	The virtual network associated with this instance.
Interfaces	The number of interfaces associated with this instance.
vRouter	The name of the vRouter associated with this instance.
IP Address	Any IP addresses associated with this instance.
Floating IP	Any floating IP addresses associated with this instance.
Traffic (In/Out)	The volume of traffic in KB or MB that is passing in and out of this instance.

Select **Monitor > Networking > Instances** to display instance traffic statistics as shown in [Figure 221 on page 604](#).

Figure 221: Instance Traffic Statistics



Monitor > Networking > Networks

Select **Monitor > Networking > Networks** to view a summary of the virtual networks in your system. See [Figure 222 on page 604](#).

Figure 222: Network Summary

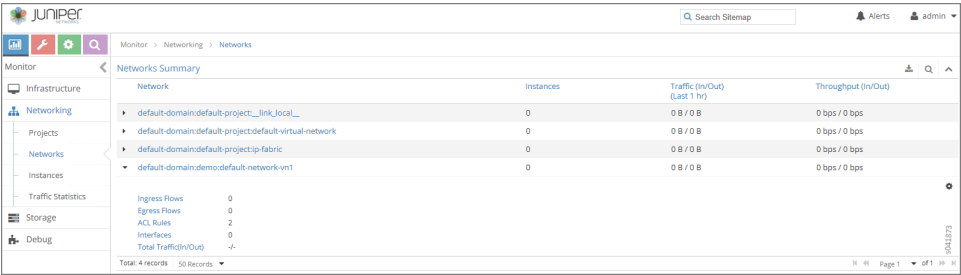


Table 84: Network Summary Fields

Field	Description
Network	The domain and network name of the virtual network. Click the arrow next to the name to display more information about the network, including the number of ingress and egress flows, the number of ACL rules, the number of interfaces, and the total traffic in and out.
Instances	The number of instances launched in this network.
Traffic (In/Out)	The volume of inter-virtual network traffic in and out of this network.
Throughput (In/Out)	The throughput of inter-virtual network traffic in and out of this network.

At **Monitor > Networking > Networks** you can click on the name of any of the listed networks to get details about the network connectivity, traffic statistics, port distribution, instances, and other details, by clicking the tabs across the top of the page.

Figure 223 on page 605 shows the **Summary** tab for an individual network, which displays connectivity details and traffic statistics for the selected network.

Figure 223: Individual Network Connectivity Details—Summary Tab

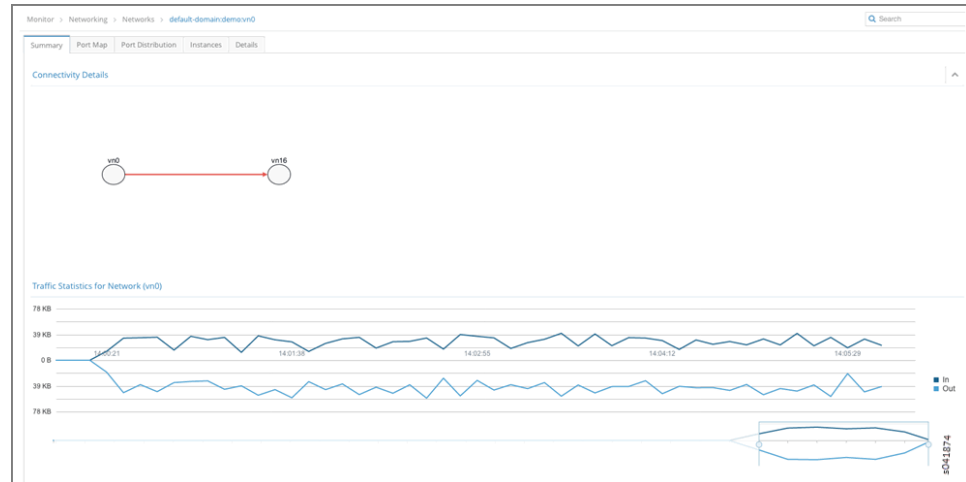


Figure 224 on page 605 shows the **Port Map** tab for an individual network, which displays the relative distribution of traffic for this network by protocol, by port.

Figure 224: Individual Network— Port Map Tab

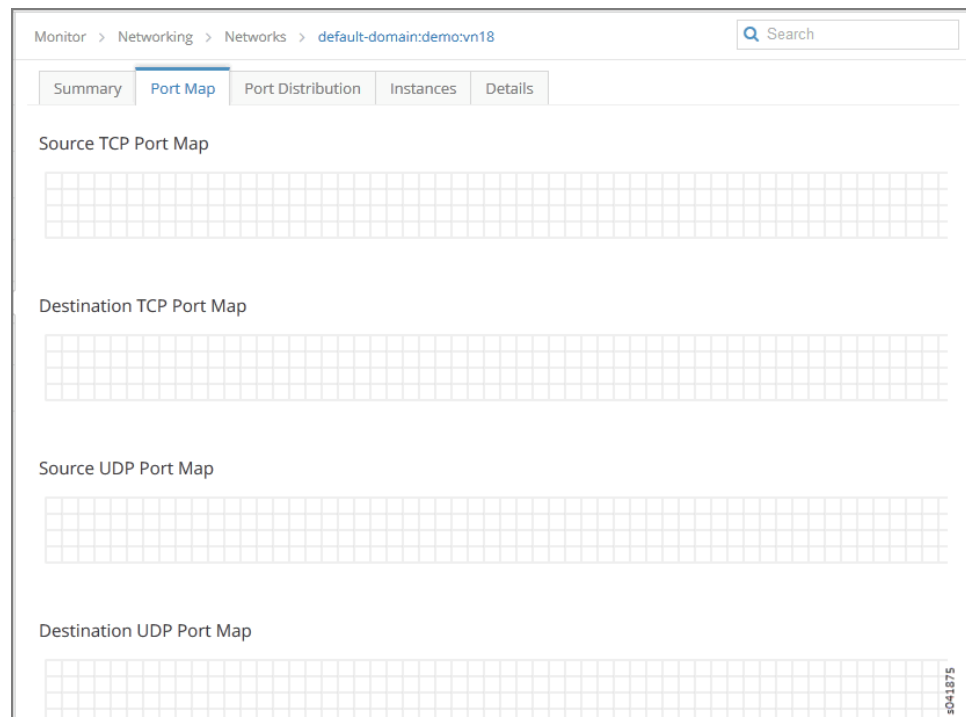


Figure 225 on page 606 shows the **Port Distribution** tab for an individual network, which displays the relative distribution of traffic in and out by source port and destination port.

Figure 225: Individual Network-- Port Distribution Tab

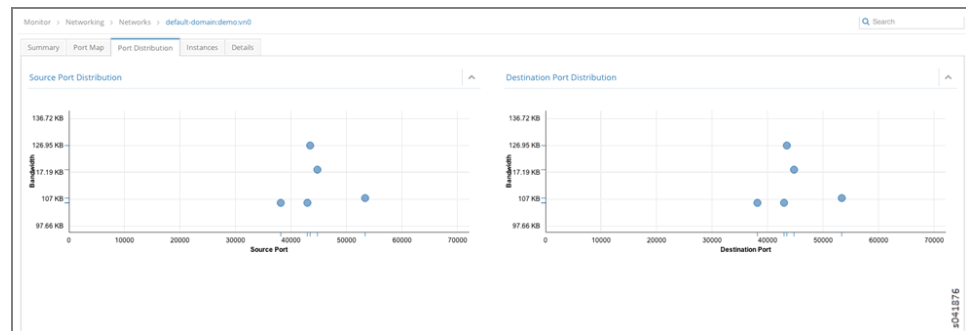


Figure 226 on page 606 shows the **Instances** tab for an individual network, which displays details for each instance associated with this network, including the number of interfaces, the associated vRouter, the instance IP address, and the volume of traffic in and out.

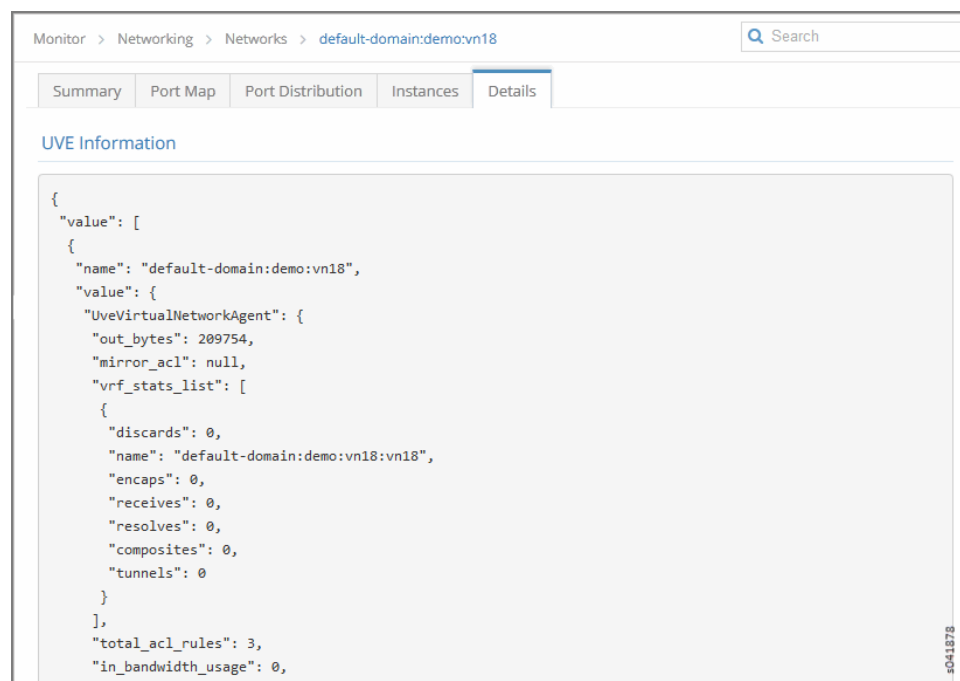
Additionally, you can click the arrow near the instance name to reveal even more details about the instance—the interfaces and their addresses, UUID, CPU (usage), and memory used of the total amount available.

Figure 226: Individual Network Instances Tab

Monitor > Networking > Networks > default-domain:demo:vn18						
Summary Port Map Port Distribution Instances Details						
Instances Summary						
	Instance	Interfaces	vRouter	IP Address	Floating IP	Traffic (In/Out)
▶	vn18_vm-b342ca93-9acd-4275-acb8-df7b5843884c	1	b1s29	192.168.18.225		1.13 KB / 712.00 B
▲	vn18_vm-22a42bf6-fccc-4db3-b5ac-80082bbefbef	1	b1s42	192.168.18.236		1.13 KB / 712.00 B
	Interfaces IP Address: 192.168.18.236 Label: 17 Mac Address: 02:e9:94:e7:0e:56 Network: default-domain:demo:vn18 Traffic (In/Out): 1.13 KB/712.00 B UUID 22a42bf6-fccc-4db3-b5ac-80082bbefbef CPU 0.01 Memory (Used/Total) 1.23 GB / 15.63 GB					
▶	vn18_vm-f676567a-826f-4e9d-9a81-b4649b7fcde2	1	b1s15	192.168.18.235		1.13 KB / 712.00 B

Figure 227 on page 607 shows the **Details** tab for an individual network, which displays the code used to define this network --the User Virtual Environment (UVE) code.

Figure 227: Individual Network Details Tab



Query > Flows

Select **Query > Flows** to perform rich and complex SQL-like queries on flows in the Contrail Controller. You can use the query results for such things as gaining insight into the operation of applications in a virtual network, performing historical analysis of flow issues, and pinpointing problem areas with flows.

- [Query > Flows > Flow Series on page 607](#)
- [Example: Query Flow Series on page 610](#)
- [Query > Flow Records on page 611](#)
- [Query > Flows > Query Queue on page 613](#)

Query > Flows > Flow Series

Select **Query > Flows > Flow Series** to create queries of the flow series table. The results are in the form of time series data for flow series. See [Figure 228 on page 608](#)

Figure 228: Query Flow Series Window

The query fields available on the screen for the **Flow Series** tab are described in [Table 85 on page 608](#). Enter query data into the fields to create a SQL-like query to display and analyze flows.

Table 85: Query Flow Series Fields

Field	Description
Time Range	<p>Select a range of time to display the flow series:</p> <ul style="list-style-type: none"> • Last 10 Mins • Last 30 Mins • Last 1 Hr • Last 6 Hrs • Last 12 Hrs • Custom <p>Click Custom to enter a specific custom time range in two fields: From Time and To Time.</p>
Select	Click the edit button (pencil icon) to open a Select window (Figure 229 on page 609), where you can click one or more boxes to select the fields to display from the flow series, such as Source VN , Dest VN , Bytes , Packets , and more.
Where	Click the edit button (pencil icon) to open a query-writing window, where you can specify query values for variables such as sourcevn , sourceip , destvn , destip , protocol , sport , dport .
Direction	Select the desired flow direction: INGRESS or EGRESS .
Filter	Click the edit button (pencil icon) to open a Filter window (Figure 230 on page 610), where you can select filter items to sort by, the sort order, and limits to the number of results returned.
Run Query	Click Run Query to retrieve the flows that match the query you created. The flows are listed on the lower portion of the screen in a box with columns identifying the selected fields for each flow.
(graph buttons)	When Time Granularity is selected, you have the option to view results in graph or flowchart form. Graph buttons appear on the screen above the Export button. Click a graph button to transform the tabular results into a graphical chart display.

Table 85: Query Flow Series Fields (*continued*)

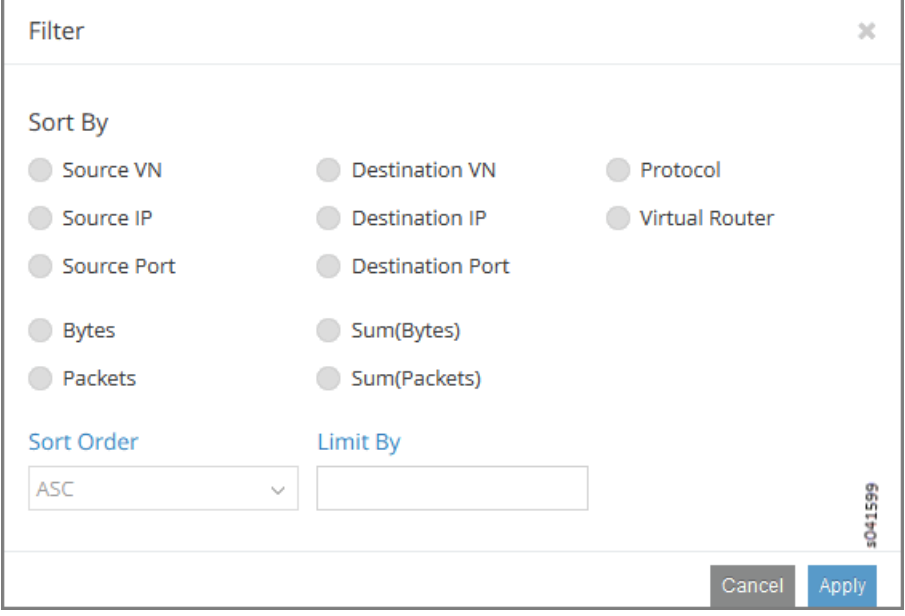
Field	Description
Export	The Export button is displayed after you click Run Query . This allows you to export the list of flows to a text .csv file.

The **Select** window allows you to select one or more attributes of a flow series by clicking the check box for each attribute desired, see [Figure 229 on page 609](#). The upper section of the **Select** window includes field names, and the lower portion lets you select units. Select **Time Granularity** and then select **SUM(Bytes)** or **SUM(Packets)** to aggregate bytes and packets in intervals.

Figure 229: Flow Series Select

Use the **Filter** window to refine the display of query results for flows, by defining an attribute by which to sort the results, the sort order of the results, and any limit needed to restrict the number of results. See [Figure 230 on page 610](#).

Figure 230: Flow Series Filter



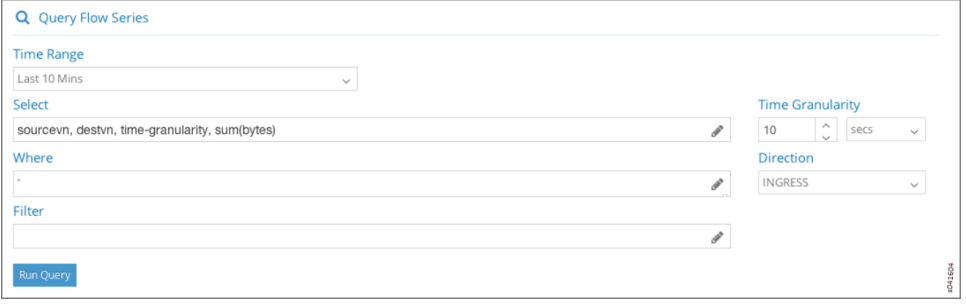
The 'Filter' dialog box contains the following elements:

- Sort By:** A grid of radio buttons for selecting the sort criteria: Source VN, Destination VN, Protocol, Source IP, Destination IP, Virtual Router, Source Port, Destination Port, Bytes, Sum(Bytes), Packets, and Sum(Packets).
- Sort Order:** A dropdown menu currently set to 'ASC'.
- Limit By:** An empty text input field.
- Buttons:** 'Cancel' and 'Apply' buttons at the bottom right.
- Footer:** A small vertical text '© 2018 Juniper Networks, Inc.' on the right side.

Example: Query Flow Series

The following is an example flow series query that returns the time series of the summation traffic in bytes for all combinations of source VN and destination VN for the last 10 minutes, with the bytes aggregated in 10 second intervals. See [Figure 231 on page 610](#).

Figure 231: Example: Query Flow Series



The 'Query Flow Series' form contains the following fields and controls:

- Time Range:** A dropdown menu set to 'Last 10 Mins'.
- Select:** A text input field containing the query: `sourcevn, destvn, time-granularity, sum(bytes)`.
- Where:** An empty text input field.
- Filter:** An empty text input field.
- Time Granularity:** A dropdown menu set to '10' with a unit dropdown set to 'secs'.
- Direction:** A dropdown menu set to 'INGRESS'.
- Buttons:** 'Run Query' button at the bottom left.
- Footer:** A small vertical text '© 2018 Juniper Networks, Inc.' on the right side.

The query returns tabular time series data, see [Figure 232 on page 611](#), for the following combinations of Source VN and Dest VN:

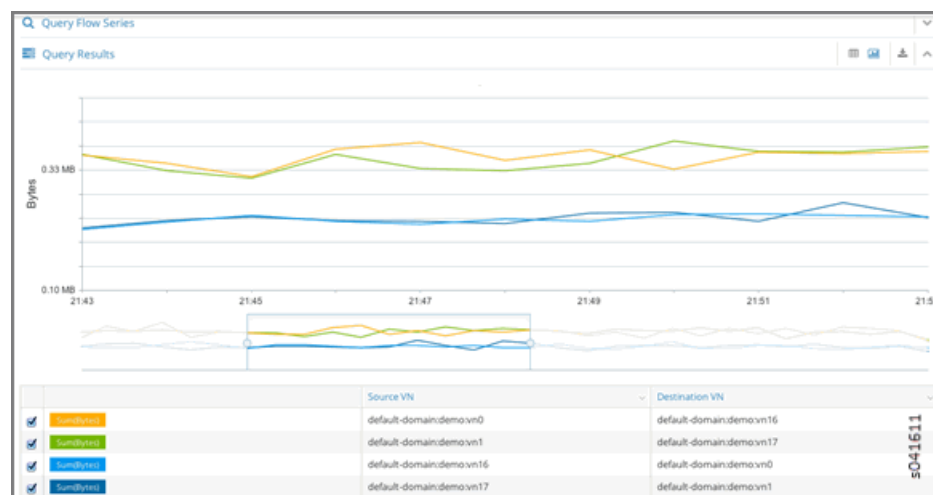
1. Flow Class 1: Source VN = default-domain:demo:front-end, Dest VN= __UNKNOWN__
2. Flow Class 2: Source VN = default-domain:demo:front-end, Dest VN=default-domain:demo:back-end

Figure 232: Query Flow Series Tabular Results

Query Flow Series				
Query Results				
Time	Source VN	Dest. VN	Direction	SUM(Bytes)
2013-08-05 18:59:30:0	default-domain:demo:vn0	default-domain:demo:vn16	INGRESS	421,128
2013-08-05 18:59:40:0	default-domain:demo:vn0	default-domain:demo:vn16	INGRESS	227,000
2013-08-05 18:59:50:0	default-domain:demo:vn0	default-domain:demo:vn16	INGRESS	216,816
2013-08-05 19:00:00:0	default-domain:demo:vn0	default-domain:demo:vn16	INGRESS	387,036
2013-08-05 18:59:30:0	default-domain:demo:vn1	default-domain:demo:vn17	INGRESS	52,944
2013-08-05 18:59:40:0	default-domain:demo:vn1	default-domain:demo:vn17	INGRESS	52,692
2013-08-05 18:59:50:0	default-domain:demo:vn1	default-domain:demo:vn17	INGRESS	58,040
2013-08-05 19:00:00:0	default-domain:demo:vn1	default-domain:demo:vn17	INGRESS	42,480
2013-08-05 18:59:30:0	default-domain:demo:vn16	default-domain:demo:vn0	INGRESS	17,832
2013-08-05 18:59:40:0	default-domain:demo:vn16	default-domain:demo:vn0	INGRESS	27,320
2013-08-05 18:59:50:0	default-domain:demo:vn16	default-domain:demo:vn0	INGRESS	20,792
2013-08-05 19:00:00:0	default-domain:demo:vn16	default-domain:demo:vn0	INGRESS	10,404

Because **Time Granularity** is selected, the results can also be displayed as graphical charts. Click the graph button on the right side of the tabular results. The results are displayed in a graphical flow chart. See [Figure 233 on page 611](#).

Figure 233: Query Flow Series Graphical Results



Query > Flow Records

Select **Query > Flow Records** to create queries of individual flow records for detailed debugging of connectivity issues between applications and virtual machines. Queries at this level return records of the active flows within a given time period.

Figure 234: Flow Records

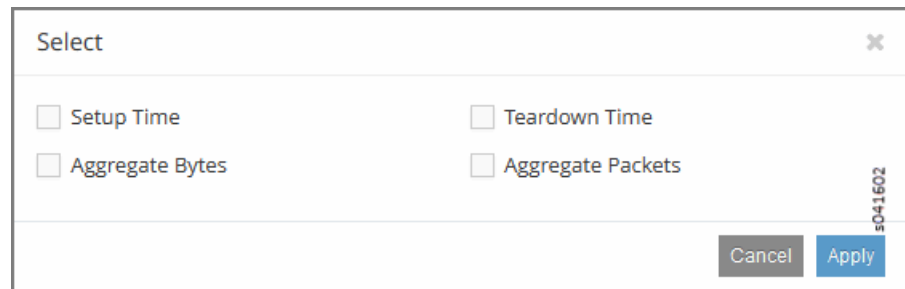
The query fields available on the screen for the **Flow Records** tab are described in [Table 86 on page 612](#). Enter query data into the fields to create an SQL-like query to display and analyze flows.

Table 86: Query Flow Records Fields

Field	Description
Time Range	<p>Select a range of time for the flow records:</p> <ul style="list-style-type: none"> • Last 10 Mins • Last 30 Mins • Last 1 Hr • Last 6 Hrs • Last 12 Hrs • Custom <p>Click Custom to enter a specified custom time range in two fields: From Time and To Time.</p>
Select	Click the edit button (pencil icon) to open a Select window (Figure 235 on page 613), where you can click one or more boxes to select attributes to display for the flow records, including Setup Time , Teardown Time , Aggregate Bytes , and Aggregate Packets .
Where	Click the edit button (pencil icon) to open a query-writing window where you can specify query values for sourcevn , sourceip , destvn , destip , protocol , sport , dport . .
Direction	Select the desired flow direction: INGRESS or EGRESS .
Run Query	Click Run Query to retrieve the flow records that match the query you created. The records are listed on the lower portion of the screen in a box with columns identifying the fields for each flow.
Export	The Export button is displayed after you click Run Query , allowing you to export the list of flows to a text .csv file.

The **Select** window allows you to select one or more attributes to display for the flow records selected, see [Figure 235 on page 613](#).

Figure 235: Flow Records Select Window



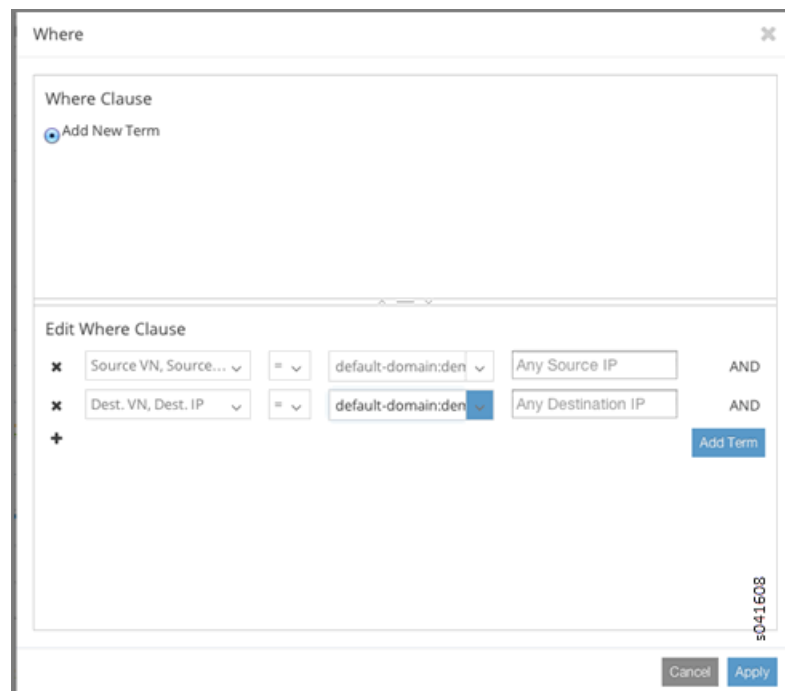
The 'Select' window contains four checkboxes: 'Setup Time', 'Teardown Time', 'Aggregate Bytes', and 'Aggregate Packets'. At the bottom right, there are 'Cancel' and 'Apply' buttons. A vertical label 's041602' is positioned on the right side of the window.

You can restrict the query to a particular source VN and destination VN combination using the **Where** section.

The **Where Clause** supports logical AND and logical OR operations, and is modeled as a logical OR of multiple AND terms. For example: ((term1 AND term2 AND term3..) OR (term4 AND term5) OR...).

Each term is a single variable expression such as **Source VN = VN1**.

Figure 236: Where Clause Window



The 'Where' window shows a 'Where Clause' section with a radio button for 'Add New Term'. Below it is an 'Edit Where Clause' section. This section contains two rows of conditions, each with a dropdown menu, an equals sign, another dropdown menu, and a text input field. The first row shows 'Source VN, Source...' followed by 'default-domain:den' and 'Any Source IP'. The second row shows 'Dest. VN, Dest. IP' followed by 'default-domain:den' and 'Any Destination IP'. Both rows are followed by an 'AND' label. At the bottom right of the 'Edit Where Clause' section is an 'Add Term' button. At the bottom of the entire window are 'Cancel' and 'Apply' buttons. A vertical label 's041608' is positioned on the right side of the window.

Query > Flows > Query Queue

Select **Query > Flows > Query Queue** to display queries that are in the queue waiting to be performed on the data. See [Figure 237 on page 614](#).

Figure 237: Flows Query Queue

Date	Query	Progress	Records	Status	Time Taken
2013-10-09 18:07:06	{ "table": "FlowSeriesTable", "start_time": 1381267020000000, "end_time": 1381277820000000, "select_fields": { "flow_class_id", "direction_ing", "sum(bytes)", "T=60", "dir": 1 } }	100%	180	completed	150 secs
2013-10-09 17:55:48	{ "table": "FlowSeriesTable", "start_time": 1381267020000000, "end_time": 1381277820000000, "select_fields": { "flow_class_id", "direction_ing", "sum(bytes)", "T=60", "dir": 1 } }	100%	180	completed	145 secs
2013-10-09 17:29:39	{ "table": "FlowSeriesTable", "start_time": 1381267020000000, "end_time": 1381277820000000, "select_fields": { "flow_class_id", "direction_ing", "sum(bytes)", "T=60", "dir": 1 } }	100%	180	completed	170 secs
2013-10-09 16:57:10	{ "table": "FlowSeriesTable", "start_time": 1381267020000000, "end_time": 1381277820000000, "select_fields": { "flow_class_id", "direction_ing", "sum(bytes)", "T=60", "dir": 1 } }	100%	180	completed	270 secs
2013-10-09 16:39:48	{ "table": "FlowSeriesTable", "start_time": 1381360140000000, "end_time": 1381361940000000, "select_fields": { "flow_class_id", "direction_ing", "T=60", "sum(bytes)", "dir": 1 } }	100%	30	completed	60 secs
2013-10-09 11:07:29	{ "table": "FlowSeriesTable", "start_time": 1381338420000000, "end_time": 1381342020000000, "select_fields": { "flow_class_id", "direction_ing", "sum(bytes)", "T=60", "dir": 1 } }	100%	7	completed	15 secs

The query fields available on the screen for the **Flow Records** tab are described in [Table 87 on page 614](#). Enter query data into the fields to create an SQL-like query to display and analyze flows.

Table 87: Query Flow Records Fields

Field	Description
Date	The date and time the query was started.
Query	A display of the parameters set for the query.
Progress	The percentage completion of the query to date.
Records	The number of records matching the query to date.
Status	The status of the query, such as completed .
Time Taken	The amount of time in seconds it has taken the query to return the matching records.
(Action icon)	Click the Action icon and select View Results to view a list of the records that match the query, or click Delete to remove the query from the queue.

See Also • [Understanding Flow Sampling on page 619](#)

Query > Logs

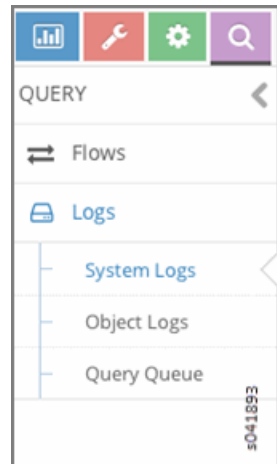
The **Query > Logs** option allows you to access the system log and object log activity of any Contrail Controller component from one central location.

- [Query > Logs Menu Options on page 615](#)
- [Query > Logs > System Logs on page 615](#)
- [Sample Query for System Logs on page 616](#)
- [Query > Logs > Object Logs on page 617](#)

Query > Logs Menu Options

Click **Query > Logs** to access the **Query Logs** menu, where you can select **System Logs** to view system log activity, **Object Logs** to view object logs activity, and **Query Queue** to create custom queries of log activity; see [Figure 238 on page 615](#).

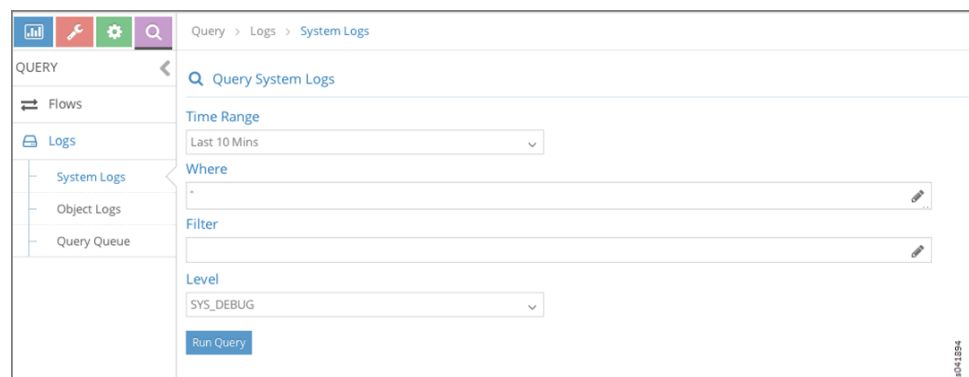
Figure 238: Query > Logs



Query > Logs > System Logs

Click **Query > Logs > System Logs** to access the **Query System Logs** menu, where you can view system logs according to criteria that you determine. See [Figure 239 on page 615](#).

Figure 239: Query > Logs > System Logs



The query fields available on the **Query System Logs** screen are described in [Table 88 on page 616](#).

Table 88: Query System Logs Fields

Field	Description
Time Range	<p>Select a range of time for which to see the system logs:</p> <ul style="list-style-type: none"> • Last 10 Mins • Last 30 Mins • Last 1 Hr • Last 6 Hrs • Last 12 Hrs • Custom <p>If you click Custom, enter a desired time range in two new fields: From Time and To Time.</p>
Where	Click the edit button (pencil icon) to open a query-writing window, where you can specify query values for variables such as Source, Module, MessageType, and the like, in order to retrieve specific information.
Level	<p>Select the message severity level to view:</p> <ul style="list-style-type: none"> • SYS_NOTICE • SYS_EMERG • SYS_ALERT • SYS_CRIT • SYS_ERR • SYS_WARN • SYS_INFO • SYS_DEBUG
Run Query	Click this button to retrieve the system logs that match the query. The logs are listed in a box with columns showing the Time , Source , Module Id , Category , Log Type , and Log message.
Export	This button appears after you click Run Query , allowing you to export the list of system messages to a text/csv file.

Sample Query for System Logs

This section shows a sample system logs query designed to show all **System Logs** from **ModuleId = VRouterAgent** on **Source = b1s16** and filtered by **Level = SYS_DEBUG**.

1. At the **Query System Logs** screen, click in the **Where** field to access the **Where** query screen and enter information defining the location to query in the **Edit Where Clause** section and click **OK**; see [Figure 240 on page 617](#).

Figure 240: Edit Where Clause

Where

Where Clause

☒ Add New Term

Edit Where Clause

✕ ModuleId = VRouterAgent AND

✕ Source = b1s16 AND

+

Add Term

OK Cancel

s041895

- The information you defined at the Where screen displays on the **Query System Logs**. Enter any more defining information needed; see [Figure 241 on page 617](#). When finished, click **Run Query** to display the results.

Figure 241: Sample Query System Logs

Q Query System Logs

Time Range

Last 10 Mins

Where

(ModuleId = VRouterAgent AND Source = b1s16)

Filter

Level

SYS_DEBUG

Run Query

s041896

Query > Logs > Object Logs

Object logs allow you to search for logs associated with a particular object, for example, all logs for a specified virtual network. Object logs record information related to modifications made to objects, including creation, deletion, and other modifications; see [Figure 242 on page 618](#).

Figure 242: Query > Logs > Object Logs

Query Object Logs

Time Range

Last 12 Hrs

Object Type

Virtual Network

Object Id

default-domain:demo:vn14

Select

ObjectLog, SystemLog

Where

*

Filter

Run Query

#041897

The query fields available on the **Object Logs** screen are described in [Table 89 on page 618](#).

Table 89: Object Logs Query Fields

Field	Description
Time Range	<div>Select a range of time for which to see the logs:</div> <div><div><div>Last 10 Mins</div><div>Last 30 Mins</div><div>Last 1 Hr</div><div>Last 6 Hrs</div><div>Last 12 Hrs</div><div>Custom</div></div><div>If you click Custom, enter a desired time range in two new fields: From Time and To Time.</div></div>
Object Type	<div>Select the object type for which to show logs:</div> <div><div><div>Virtual Network</div><div>Virtual Machine</div><div>Virtual Router</div><div>BGP Peer</div><div>Routing Instance</div><div>XMPP Connection</div></div></div>
Object Id	<div>Select from a list of available identifiers the name of the object you wish to use.</div>
Select	<div>Click the edit button (pencil icon) to open a window where you can select searchable types by clicking a checkbox:</div> <div><div><div>ObjectLog</div><div>SystemLog</div></div></div>

Table 89: Object Logs Query Fields (*continued*)

Field	Description
Where	Click the edit button (pencil icon) to open the query-writing window, where you can specify query values for variables such as Source , ModuleId , and MessageType , in order to retrieve information as specific as you wish.
Run Query	Click this button to retrieve the system logs that match the query. The logs are listed in a box with columns showing the Time , Source , Module Id , Category , Log Type , and Log message.
Export	This button appears after you click Run Query , allowing you to export the list of system messages to a text/csv file.

Understanding Flow Sampling

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

- [Flow Sampling on page 619](#)
- [Flow Handling on page 620](#)
- [Flow Aging on page 620](#)
- [TCP State-Based Flow Handling and Aging on page 621](#)

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.



NOTE: In releases earlier than Contrail Release 2.22, all flow records were exported from the agent. Depending on the scale of flows, some of these exported flows might be dropped due to queue overflow.

In Contrail Release 2.22 and later, flow records are sampled and exported to the Contrail Collector based on the sampling.

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. In releases earlier than Contrail Release 2.22, the overflow table was a global table, which is searched sequentially. In Contrail Release 2.22 and later, the overflow entries are maintained 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 on page 621](#)
- [Protocol Based Flow Aging on page 621](#)
- [Fat Flow on page 622](#)

TCP State-Based Flow Handling

In Contrail Release 2.22 and later, the Contrail vRouter 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 supported in Contrail Release 2.22 and later.

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.**

Fat Flow

In Contrail Release 2.22 and later, Contrail supports optimization to reduce the number of flows set up by reusing a flow. Consequently, a single flow pair can be used for any number of sessions between two endpoints for the same application protocol.

Any number of DNS sessions from a client to the server can use a single flow pair. The effect is that the flow hash key is reduced from five-tuples to four-tuples consisting of source and destination IP addresses, the server port, and the IP protocol. The client port is not used in the flow key.

This feature can be configured 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. In the example, the server **virtual-machine-interface**, protocol **udp** and port **53** can be configured as a fat-flow-protocol.

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.

Related Documentation

- [Query > Flows on page 607](#)

Example: Debugging Connectivity Using Monitoring for Troubleshooting

- [Using Monitoring to Debug Connectivity on page 622](#)

Using Monitoring to Debug Connectivity

This example shows how you can use monitoring to debug connectivity in your Contrail system. You can use the demo setup in Contrail to use these steps on your own.

1. Navigate to **Monitor -> Networking -> Networks -> default-domain:demo:vn0, Instance ed6abd16-250e-4ec5-a382-5cbc458fb0ca** with IP address **192.168.0.252** in the virtual network **vn0**; see [Figure 243 on page 622](#)

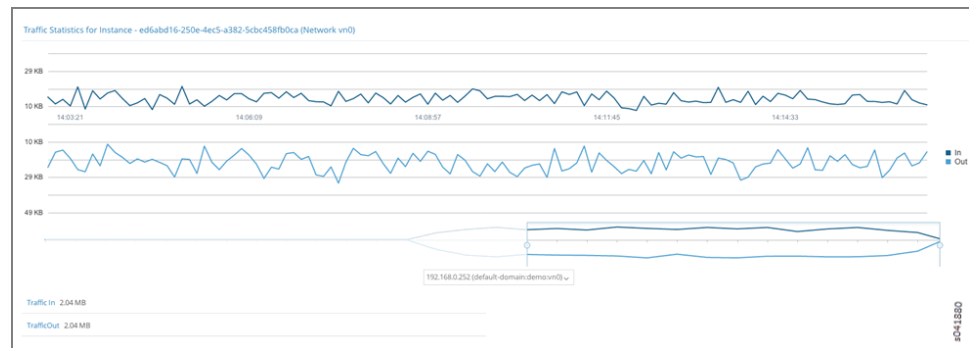
Figure 243: Navigate to Instance



Instance	Traffic In	Traffic Out
ed6abd16-250e-4ec5-a382-5cbc458fb0ca	1.73 MB	1.74 MB
682b7414-c8ba-45ee-9f8c-9c22bd8dc69d	1.72 MB	1.72 MB

2. Click the instance to view **Traffic Statistics for Instance**. see [Figure 244 on page 623](#).

Figure 244: Traffic Statistics for Instance

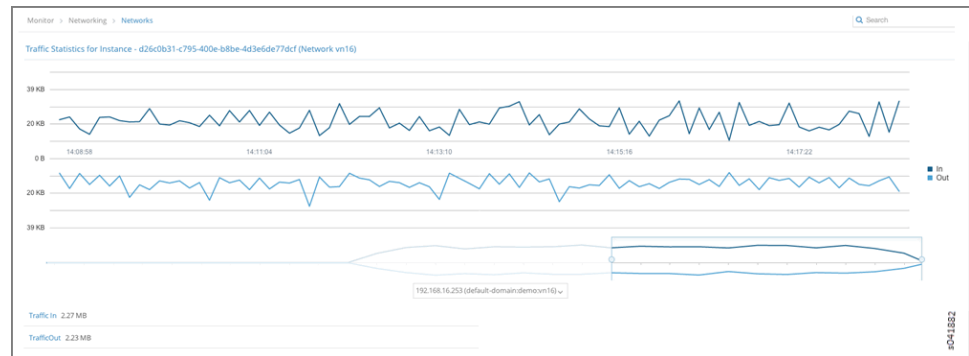


3. Instance d26c0b31-c795-400e-b8be-4d3e6de77dcf with IP address 192.168.0.253 in the virtual network **vn16**. see [Figure 245 on page 623](#) and [Figure 246 on page 623](#).

Figure 245: Navigate to Instance

Instance	Traffic In	Traffic Out
d26c0b31-c795-400e-b8be-4d3e6de77dcf	2.18 MB	2.13 MB
23045415-b679-4d8a-8f9d-96c162dc28be	2.11 MB	2.16 MB

Figure 246: Traffic Statistics for Instance



4. From **Monitor->Infrastructure->Virtual Routers->a3s18->Interfaces**, we can see that Instance ed6abd16-250e-4ec5-a382-5cbc458fb0ca is hosted on Virtual Router **a3s18**; see [Figure 247 on page 623](#).

Figure 247: Navigate to a3s18 Interfaces

Name	Label	Status	Network	IP Address	Floating IP	Instance
tap1daed0121-4c	16	Up	default-domain:demo:vn0	192.168.0.252	None	ed6abd16-250e-4ec5-a382-5cbc458fb0ca
tap2496a2e1-97	18	Up	default-domain:demo:vn16	192.168.16.252	None	23045415-b679-4d8a-8f9d-96c162dc28be
tap5b3b3d63-74	19	Up	default-domain:demo:vn17	192.168.17.252	None	99311eda-261e-47e8-b4d7-8d126d7d99df
tapc7d0843c-6b	17	Up	default-domain:demo:vn1	192.168.1.252	None	20244ef9-a4ed-4a32-8038-15c5323572e

- From **Monitor->Infrastructure->Virtual Routers->a3s19->Interfaces**, we can see that Instance **d26c0b31-c795-400e-b8be-4d3e6de77dcf** is hosted on Virtual Router **a3s19**; see [Figure 248 on page 624](#).

Figure 248: Navigate to a3s19 Interfaces

Monitor > Infrastructure > Virtual Routers > a3s19							Search
Details	Console	Interfaces	Networks	ACL	Flows	Routes	
Name	Label	Status	Network	IP Address	Floating IP	Instance	
tap2958592f42	19	Up	default-domain:demo:vn16	192.168.16.253	None	d26c0b31-c795-400e-b8be-4d3e6de77dcf	e0418354
tapb257d164d3	18	Up	default-domain:demo:vn1	192.168.1.253	None	eebc321-7536-46e7-a454-cdf1f13ac995	
tapc3e3d98746	17	Up	default-domain:demo:vn17	192.168.17.253	None	b242595-67e-4b60-9478-81a4a829541	
tapfeaf97e355	16	Up	default-domain:demo:vn0	192.168.0.253	None	682b7414-c4ba-45ee-91bc-9c22cdd6c1dd	

- Virtual Routers **a3s18** and **a3s19** have the ACL entries to allow connectivity between **default-domain:demo:vn0** and **default-domain:demo:vn16** networks; see [Figure 249 on page 624](#) and [Figure 250 on page 624](#).

Figure 249: ACL Connectivity a3s18

Monitor > Infrastructure > Virtual Routers > a3s18											Search
Details	Console	Interfaces	Networks	ACL	Flows	Routes					
UUID	Flows	Action	Protocol	Source Network or Prefix	Source Port	Destination Network or Prefix	Destination Port	Source Policy Rule	ACE Id		
a726326e-3f50-477a-ad...	16	pass	any	default-domain:demo:vn0	any	default-domain:demo:vn16	any		1	e0418355	
		pass	any	default-domain:demo:vn16	any	default-domain:demo:vn0	any		2		
		pass	any	default-domain:demo:vn0	any	default-domain:demo:vn0	any		3		
b32143a3-0e80-4aa2-9c...	16	pass	any	default-domain:demo:vn1	any	default-domain:demo:vn17	any		1		
		pass	any	default-domain:demo:vn17	any	default-domain:demo:vn1	any		2		
		pass	any	default-domain:demo:vn1	any	default-domain:demo:vn1	any		3		
b8c79810-ef9c-418a-a7...	16	pass	any	default-domain:demo:vn0	any	default-domain:demo:vn16	any		1		
		pass	any	default-domain:demo:vn16	any	default-domain:demo:vn0	any		2		
		pass	any	default-domain:demo:vn16	any	default-domain:demo:vn16	any		3		
d7b47291-7a21-495e-8d...	16	pass	any	default-domain:demo:vn1	any	default-domain:demo:vn17	any		1		
		pass	any	default-domain:demo:vn17	any	default-domain:demo:vn1	any		2		
		pass	any	default-domain:demo:vn17	any	default-domain:demo:vn17	any		3		

Figure 250: ACL Connectivity a3s19

Monitor > Infrastructure > Virtual Routers > a3s19											Search
Details	Console	Interfaces	Networks	ACL	Flows	Routes					
UUID	Flows	Action	Protocol	Source Network or Prefix	Source Port	Destination Network or Prefix	Destination Port	Source Policy Rule	ACE Id		
a726326e-3f50-477a-ad...	16	pass	any	default-domain:demo:vn0	any	default-domain:demo:vn16	any		1	e0418356	
		pass	any	default-domain:demo:vn16	any	default-domain:demo:vn0	any		2		
		pass	any	default-domain:demo:vn0	any	default-domain:demo:vn0	any		3		
b32143a3-0e80-4aa2-9c...	16	pass	any	default-domain:demo:vn1	any	default-domain:demo:vn17	any		1		
		pass	any	default-domain:demo:vn17	any	default-domain:demo:vn1	any		2		
		pass	any	default-domain:demo:vn1	any	default-domain:demo:vn1	any		3		
b8c79810-ef9c-418a-a7...	16	pass	any	default-domain:demo:vn0	any	default-domain:demo:vn16	any		1		
		pass	any	default-domain:demo:vn16	any	default-domain:demo:vn0	any		2		
		pass	any	default-domain:demo:vn16	any	default-domain:demo:vn16	any		3		
d7b47291-7a21-495e-8d...	16	pass	any	default-domain:demo:vn1	any	default-domain:demo:vn17	any		1		
		pass	any	default-domain:demo:vn17	any	default-domain:demo:vn1	any		2		
		pass	any	default-domain:demo:vn17	any	default-domain:demo:vn17	any		3		

- Next, verify the routes on the control node for routing instances **default-domain:demo:vn0:vn0** and **default-domain:demo:vn16:vn16**; see [Figure 251 on page 625](#) and [Figure 252 on page 625](#).

Figure 251: Routes default-domain:demo:vn0:vn0

Monitor > Infrastructure > Control Nodes > a3s15							
<div> <div>Details</div> <div>Console</div> <div>Peers</div> <div>Routes</div> </div>							
<div> <div>Routing Instance</div> <div>default-domain:demo</div> </div>		<div> <div>Address Family</div> <div>All</div> </div>		<div> <div>Limit 50 Routes</div> </div>			
<div> <div>Peer Source</div> <div>All</div> </div>		<div> <div>Prefix</div> <div>Prefix</div> </div>		<div> <div>Display Routes</div> <div>Reset</div> </div>			
Prefix	Address Family	Protocol	Source	Next hop	Label	Local Preference	AS Path
192.168.0.252/32	inet	XMPP	a3s18	10.84.17.4	16	100	-
	inet	BGP	10.84.17.3	10.84.17.4	16	100	AS_PATH: 0
192.168.0.253/32	inet	XMPP	a3s19	10.84.17.5	16	100	-
	inet	BGP	10.84.17.3	10.84.17.5	16	100	AS_PATH: 0
192.168.16.252/32	inet	XMPP	a3s18	10.84.17.4	17	100	-
	inet	BGP	10.84.17.3	10.84.17.4	17	100	AS_PATH: 0
192.168.16.253/32	inet	XMPP	a3s19	10.84.17.5	17	100	-
	inet	BGP	10.84.17.3	10.84.17.5	17	100	AS_PATH: 0
10.84.17.4:1:192.168.0.255,0.0.0.0	inetmcast	XMPP	a3s18	10.84.17.4	0	100	-
10.84.17.4:1:255.255.255.255,0.0.0.0	inetmcast	XMPP	a3s18	10.84.17.4	0	100	-
10.84.17.5:1:192.168.0.255,0.0.0.0	inetmcast	XMPP	a3s19	10.84.17.5	0	100	-
10.84.17.5:1:255.255.255.255,0.0.0.0	inetmcast	XMPP	a3s19	10.84.17.5	0	100	-

Figure 252: Routes default-domain:demo:vn16:vn16

Monitor > Infrastructure > Control Nodes > a3s15							
<div> <div>Details</div> <div>Console</div> <div>Peers</div> <div>Routes</div> </div>							
<div> <div>Routing Instance</div> <div>default-domain:demo</div> </div>		<div> <div>Address Family</div> <div>All</div> </div>		<div> <div>Limit 50 Routes</div> </div>			
<div> <div>Peer Source</div> <div>All</div> </div>		<div> <div>Prefix</div> <div>Prefix</div> </div>		<div> <div>Display Routes</div> <div>Reset</div> </div>			
Prefix	Address Family	Protocol	Source	Next hop	Label	Local Preference	AS Path
192.168.0.252/32	inet	XMPP	a3s18	10.84.17.4	16	100	-
	inet	BGP	10.84.17.3	10.84.17.4	16	100	AS_PATH: 0
192.168.0.253/32	inet	XMPP	a3s19	10.84.17.5	16	100	-
	inet	BGP	10.84.17.3	10.84.17.5	16	100	AS_PATH: 0
192.168.16.252/32	inet	XMPP	a3s18	10.84.17.4	17	100	-
	inet	BGP	10.84.17.3	10.84.17.4	17	100	AS_PATH: 0
192.168.16.253/32	inet	XMPP	a3s19	10.84.17.5	17	100	-
	inet	BGP	10.84.17.3	10.84.17.5	17	100	AS_PATH: 0
10.84.17.4:2:192.168.16.255,0.0.0.0	inetmcast	XMPP	a3s18	10.84.17.4	0	100	-
10.84.17.4:2:255.255.255.255,0.0.0.0	inetmcast	XMPP	a3s18	10.84.17.4	0	100	-
10.84.17.5:2:192.168.16.255,0.0.0.0	inetmcast	XMPP	a3s19	10.84.17.5	0	100	-
10.84.17.5:2:255.255.255.255,0.0.0.0	inetmcast	XMPP	a3s19	10.84.17.5	0	100	-

8. We can see that VRF **default-domain:demo:vn0:vn0** on Virtual Router **a3s18** has the appropriate route and next hop to reach VRF **default-domain:demo:front-end** on Virtual Router **a3s19**; see [Figure 253 on page 626](#).

Figure 253: Verify Route and Next Hop a3s18

Monitor > Infrastructure > Virtual Routers > a3s18		
Details	Console	Interfaces
Networks	ACL	Flows
Routes		
VRF	default-domain:demo:vn0:vn0	Show Routes <input checked="" type="radio"/> Unicast <input type="radio"/> Multicast
Prefix	Next ho...	Next hop details
169.254.169.254 / 32	receive	Source: MData Dest VN: default-domain:default-project:link_local__
192.168.0.252 / 32	interface	Interface: tap1dae0121-4c Dest VN: default-domain:demo:vn0
	interface	Interface: tap1dae0121-4c Dest VN: default-domain:demo:vn0
	interface	Interface: tap1dae0121-4c Dest VN: default-domain:demo:vn0
192.168.0.253 / 32	tunnel	Dest IP: 10.84.17.5 Dest VN: default-domain:demo:vn0 Label: 16
	tunnel	Dest IP: 10.84.17.5 Dest VN: default-domain:demo:vn0 Label: 16
192.168.0.254 / 32	interface	Interface: pkt0 Dest VN: default-domain:demo:vn0
192.168.16.252 / 32	interface	Interface: tap249de2e1-97 Dest VN: default-domain:demo:vn16
	interface	Interface: tap249de2e1-97 Dest VN: default-domain:demo:vn16
192.168.16.253 / 32	tunnel	Dest IP: 10.84.17.5 Dest VN: default-domain:demo:vn16 Label: 19

9. We can see that VRF **default-domain:demo:vn16:vn16** on Virtual Router **a3s19** has the appropriate route and next hop to reach VRF **default-domain:demo:vn0:vn0** on Virtual Router **a3s18**; see [Figure 254 on page 626](#).

Figure 254: Verify Route and Next Hop a3s19

Monitor > Infrastructure > Virtual Routers > a3s19		
Details	Console	Interfaces
Networks	ACL	Flows
Routes		
VRF	default-domain:demo:vn16:vn16	Show Routes <input checked="" type="radio"/> Unicast <input type="radio"/> Multicast
Prefix	Next ho...	Next hop details
169.254.169.254 / 32	receive	Source: MData Dest VN: default-domain:default-project:link_local__
192.168.0.252 / 32	tunnel	Dest IP: 10.84.17.4 Dest VN: default-domain:demo:vn0 Label: 16
	tunnel	Dest IP: 10.84.17.4 Dest VN: default-domain:demo:vn0 Label: 16
192.168.0.253 / 32	interface	Interface: tape5ea97e3-55 Dest VN: default-domain:demo:vn0
	interface	Interface: tape5ea97e3-55 Dest VN: default-domain:demo:vn0
192.168.16.252 / 32	tunnel	Dest IP: 10.84.17.4 Dest VN: default-domain:demo:vn16 Label: 18
	tunnel	Dest IP: 10.84.17.4 Dest VN: default-domain:demo:vn16 Label: 18
192.168.16.253 / 32	interface	Interface: tap29585b2f-c2 Dest VN: default-domain:demo:vn16
	interface	Interface: tap29585b2f-c2 Dest VN: default-domain:demo:vn16
	interface	Interface: tap29585b2f-c2 Dest VN: default-domain:demo:vn16
192.168.16.254 / 32	interface	Interface: pkt0 Dest VN: default-domain:demo:vn16

10. Finally, flows between instances (IPs **192.168.0.252** and **192.168.16.253**) can be verified on Virtual Routers **a3s18** and **a3s19**; see [Figure 255 on page 627](#) and [Figure 256 on page 627](#).

Figure 255: Flows for a3s18

Monitor > Infrastructure > Virtual Routers > a3s18

Details Console Interfaces Networks ACL Flows Routes

Active Flows: 64

Protocol	Source Network	Source IP	Source Port	Destination Network	Destination IP	Destination Port	Bytes/Pkts	Setup Time
TCP	vn0	192.168.0.252	43434	vn16	192.168.16.253	9100	1884588/5417	21:00:22.131180 2013-Aug-06
TCP	vn16	192.168.16.253	9100	vn0	192.168.0.252	43434	1969658/5891	21:00:22.131193 2013-Aug-06
TCP	vn16	192.168.16.253	9101	vn0	192.168.0.252	53369	1903050/5805	21:00:22.206222 2013-Aug-06
TCP	vn0	192.168.0.252	53369	vn16	192.168.16.253	9101	1890088/5302	21:00:22.206287 2013-Aug-06
UDP	vn0	192.168.0.252	39522	vn16	192.168.16.252	9200	0/0	21:00:22.382861 2013-Aug-06
UDP	vn0	192.168.0.252	44794	vn16	192.168.16.253	9201	1707392/3144	21:00:24.104277 2013-Aug-06
UDP	vn16	192.168.16.253	9201	vn0	192.168.0.252	44794	1785789/3107	21:00:24.104293 2013-Aug-06
UDP	vn0	192.168.0.252	40561	vn16	192.168.16.253	9200	1659476/3067	21:00:22.037377 2013-Aug-06
UDP	vn16	192.168.16.253	9200	vn0	192.168.0.252	40561	1643324/3061	21:00:22.037387 2013-Aug-06
UDP	vn0	192.168.0.252	39522	vn16	192.168.16.252	9200	1676616/3074	21:00:22.306703 2013-Aug-06
TCP	vn0	192.168.0.252	34236	vn16	192.168.16.252	9100	1891368/5486	21:00:22.395695 2013-Aug-06
TCP	vn0	192.168.0.252	34236	vn16	192.168.16.252	9100	0/0	21:00:22.400371 2013-Aug-06

Figure 256: Flows for a3s19

Monitor > Infrastructure > Virtual Routers > a3s19

Details Console Interfaces Networks ACL Flows Routes

Active Flows: 64

Protocol	Source Network	Source IP	Source Port	Destination Network	Destination IP	Destination Port	Bytes/Pkts	Setup Time
UDP	vn0	192.168.0.252	44794	vn16	192.168.16.253	9201	1049880/1975	21:00:24.111324 2013-Aug-06
UDP	vn16	192.168.16.253	9201	vn0	192.168.0.252	44794	1109004/1963	21:00:24.111380 2013-Aug-06
UDP	vn0	192.168.0.252	40561	vn16	192.168.16.253	9200	1046756/1877	21:00:22.047747 2013-Aug-06
UDP	vn16	192.168.16.253	9200	vn0	192.168.0.252	47270	1061900/1921	21:00:25.373941 2013-Aug-06
UDP	vn16	192.168.16.253	9200	vn0	192.168.0.252	40561	1010568/1914	21:00:22.047756 2013-Aug-06
TCP	vn16	192.168.16.253	9100	vn0	192.168.0.253	53314	1217772/3649	21:00:23.440564 2013-Aug-06
TCP	vn0	192.168.0.252	43434	vn16	192.168.16.253	9100	1196536/3400	21:00:22.137665 2013-Aug-06
TCP	vn16	192.168.16.253	9100	vn0	192.168.0.252	43434	1239616/3724	21:00:22.137679 2013-Aug-06
UDP	vn16	192.168.16.253	9200	vn0	192.168.0.253	47270	0/0	21:00:25.347868 2013-Aug-06
TCP	vn16	192.168.16.253	9100	vn0	192.168.0.253	53314	0/0	21:00:23.440990 2013-Aug-06
UDP	vn16	192.168.16.253	9201	vn0	192.168.0.253	53930	1088692/1953	21:00:25.443166 2013-Aug-06
UDP	vn16	192.168.16.253	9101	vn0	192.168.0.253	34551	0/0	21:00:23.514246 2013-Aug-06
TCP	vn16	192.168.16.253	9101	vn0	192.168.0.253	34551	1394273/3004	21:00:23.513463 2013-Aug-06

CHAPTER 26

Common Support Answers

- [Debugging Ping Failures for Policy-Connected Networks on page 629](#)
- [Debugging BGP Peering and Route Exchange in Contrail on page 635](#)
- [Troubleshooting the Floating IP Address Pool in Contrail on page 651](#)
- [Removing Stale Virtual Machines and Virtual Machine Interfaces on page 674](#)
- [Troubleshooting Link-Local Services in Contrail on page 678](#)

Debugging Ping Failures for Policy-Connected Networks

This topic presents troubleshooting scenarios and steps for resolving reachability issues (ping failures) when working with policy-connected virtual networks.

These are the methods used to configure reachability for a virtual network or virtual machine:

- Use network policy to exchange virtual network routes.
- Use a floating IP address pool to associate an IP address from a destination virtual network to virtual machine(s) in the source virtual network.
- Use an ASN/RT configuration to exchange virtual network routes with an MX Series router gateway.
- Use a service instance static route configuration to route between service instances in two virtual networks.

This topic focuses on troubleshooting reachability for the first method --- using network policy to exchange routes between virtual networks.

Troubleshooting Procedure for Policy-Connected Network

1. Check the state of the virtual machine and interface.

Before doing anything else, check the status of the source and destination virtual machines.

- Is the **Status** of each virtual machine **Up**?
- Are the corresponding tap interfaces **Active**?

Check the virtual machine status in the Contrail UI:

Figure 257: Virtual Machine Status Window

Name	Label	Status	Network	IP Address	Floating IP	Instance
tap808d9c6-67	16	Up	vn1 (admin)	31.1.1.253	10.204.219.108	549533ef-403e-403e-bc47-6c748e6f5c8 / vn1

Check the tap interface status in the http agent introspect, for example:

http://nodef1.englab.juniper.net:8085/Snh_ItfReq?name=

Figure 258: Tap Interface Status Window

Index	name	uuid	vrf_name	active
4	tap808d9c6-67	888d9c6-672e-4c1e-9e83-e053d6c911ab	default-domain:admin:vn1:vn1	Active

When the virtual machine status is verified **Up**, and the tap interface is **Active**, you can focus on other factors that affect traffic, including routing, network policy, security policy, and service instances with static routes.

2. Check reachability and routing.

Use the following troubleshooting guidelines whenever you are experiencing ping failures on virtual network routes that are connected by means of network policy.

Check the network policy configuration:

- Verify that the policy is attached to each of the virtual networks.
- Each attached policy should have either an explicit rule allowing traffic from one virtual network to the other, or an allow all traffic rule.
- Verify that the order of the actions in the policy rules is correct, because the actions are applied in the order in which they are listed.
- If there are multiple policies attached to a virtual network, verify that the policies are attached in a logical order. The first policy listed is applied first, and its rules are applied first, then the next policy is applied.
- Finally, if either of the virtual networks does not have an explicit rule to allow traffic from the other virtual network, the traffic flow will be treated as an **UNRESOLVED** or **SHORT** flow and all packets will be dropped.

Use the following sequence in the Contrail UI to check policies, attachments, and traffic rules:

Check VNI-VN2 ACL information from the compute node:

Figure 259: Policies, Attachments, and Traffic Rule Status Window

UUID	Flows	Action	Protocol	Source Network or Prefix	Source Port	Destination Network or Prefix	Destination Port	ACE Id
8b0329d7-ad9e-41ac-a72e-30f4dbc2b5ee	100000	pass	1-1	default-domainadminvn1	any	default-domainadminvn2	any	1
		pass	1-1	default-domainadminvn2	any	default-domainadminvn1	any	2
		pass	any	default-domainadminvn1	any	default-domainadminvn1	any	3
		deny	any	default-domainadminvn1	any	default-domainadminvn2	any	4
		deny	any	default-domainadminvn2	any	default-domainadminvn1	any	5

Check the virtual network policy configuration with route information:

Figure 260: Virtual Network Policy Configuration Window

Network	Attached Policies	IP Blocks
vn1	default-analyzer-analyzer-policy vn1-vn2	31.1.1.0/24
vn2	allow_all	32.1.1.0/24

Check the VN1 route information for VN2 routes:

Figure 261: Virtual Network Route Information Window

Prefix	Next hop Type	Next hop details
32.1.1.251 / 32	tunnel	Source IP: 192.168.40.11 Destination IP: 192.168.40.11 disabled Valid: true
32.1.1.252 / 32	interface	Interface: tap2e0a5c4d-1f Destination VN: default-domain:admin:vn1:vn2
32.1.1.253 / 32	tunnel	Source IP: 192.168.40.11 Destination IP: 192.168.40.11 disabled Valid: true

If a route is missing, ping fails. Flow inspection in the compute node displays **Action: D(rop)**.

Repeated dropstats commands confirms the drop by incrementing the **Flow Action Drop** counter with each iteration of dropstats.

Flow and dropstats commands issued at the compute node:

Figure 262: Flow and Dropstats Command List

```

root@nodefl:~# flow -l | grep 32.1.1 -Al
root@nodefl:~# flow -l | grep 32.1.1 -Al
root@nodefl:~# flow -l | grep 32.1.1 -Al
73348          32.1.1.252:1911          31.1.1.253:0          1 (1)
                (Action:D, S(nh):0, Statistics:0/0)
--
423404          31.1.1.253:1911          32.1.1.252:0          1 (1)
                (Action:D, S(nh):8, Statistics:1/84)
root@nodefl:~# dropstats | grep "Flow Action Drop "
Flow Action Drop          1588
root@nodefl:~# dropstats | grep "Flow Action Drop "
Flow Action Drop          1589
root@nodefl:~# dropstats | grep "Flow Action Drop "
Flow Action Drop          1590
root@nodefl:~#

```

To help in debugging flows, you can use the detailed flow query from the agent introspect page for the compute node.

Fields of interest include:

- Inputs [from **flow -l** output]: **src/dest ip**, **src/dest ports**, **protocol**, and **vrf**
- Output from detailed flow query: **short_flow**, **src_vn**, **action_str->action**

Flow command output:

Figure 263: Flow Command Output Window

[screen 8: tcsh] — ssh — 171x30			
Flow table			
Index	Source:Port	Destination:Port	Proto(V)
0	31.1.1.253:18572 (Action:D, S(nh):8, Statistics:4447/204562 Mirror Index : 0)	32.1.1.252:9	17 (1)
1	32.1.1.252:9 (Action:D, S(nh):15, Statistics:0/0 Mirror Index : 0)	31.1.1.253:34318	17 (1)
4	31.1.1.253:4391 (Action:D, S(nh):8, Statistics:4447/204562 Mirror Index : 0)	32.1.1.252:9	17 (1)
5	31.1.1.253:34163 (Action:D, S(nh):8, Statistics:4446/204516 Mirror Index : 0)	32.1.1.252:9	17 (1)

Fetching details of a single flow:

Figure 264: Fetch Flow Record Window

Output from **FetchFlowRecord** shows unresolved IP addresses:

Figure 265: Unresolved IP Address Window

implicit_deny	no
short_flow	yes
setup_time_utc	1394959054698162
local_flow	no
src_vn	__UNKNOWN__
dst_vn	__UNKNOWN__
reverse_flow	no

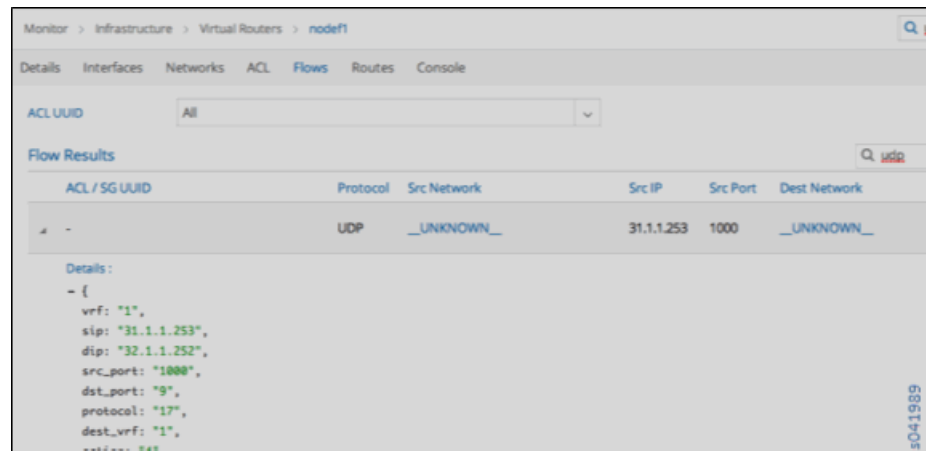
```

    },
    action_str: - {
      list: - {
        ActionStr: - {
          action: "drop"
        }
      }
    }
  }

```

You can also retrieve information about unresolved flows from the Contrail UI, as shown in the following:

Figure 266: Unresolved Flow Details Window

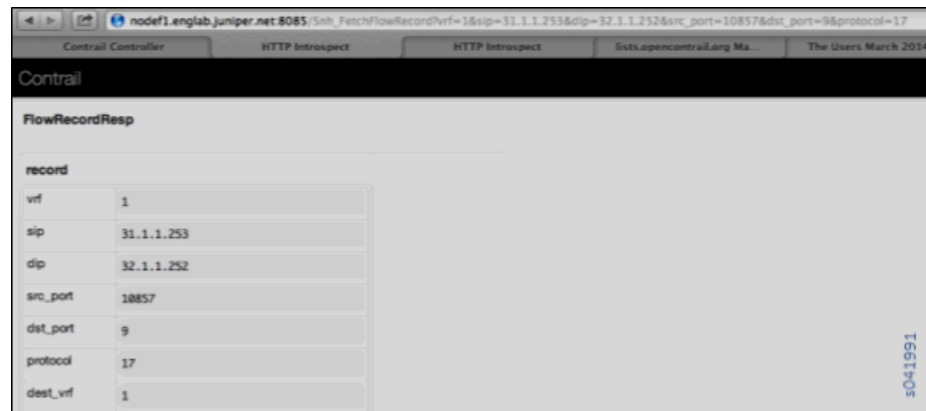


3. Check for protocol-specific network policy action.

If you are still experiencing reachability issues, troubleshoot any protocol-specific action, where routes are exchanged, but only specific protocols are allowed.

The following shows a sample query on a protocol-specific flow in the agent introspect:

Figure 267: Protocol-Specific Flow Sample



The following shows that although the virtual networks are resolved (not __UNKNOWN__), and not a short flow (the flow entry exists for a defined aging time), the policy action clearly displays **deny** as the action.

Figure 268: Protocol-Specific Flow Sample With Deny Action

implicit_deny	no
short_flow	no
setup_time_utc	1394972834710415
local_flow	no
src_vn	default-domain:admin:vn1
dst_vn	default-domain:admin:vn2
reverse_flow	no
policy	<div> <div>policy</div> <div> <div>action</div> <div>g</div> </div> <div> <div>acl</div> <div> <div>uuid</div> <div>8b0329d7-ad9e-41ac-af2e-30f4dbc2b50e</div> </div> </div> <div> <div>action_str</div> <div> <div>action</div> <div>deny</div> </div> </div> </div>

Summary

This topic explores one area —debugging for policy-based routing. However, in a complex system, a virtual network might have one or more configuration methods combined that influence reachability and routing.

For example, an environment might have a virtual network VN-X configured with policy-based routing to another virtual network VN-Y. At the same time, there are a few virtual machines in VN-X that have a floating IP to another virtual network VN-Z, which is connected to VN-XX via a NAT service instance. This is a complex scenario, and you need to debug step-by-step, taking into account all of the features working together.

Additionally, there are other considerations beyond routing and reachability that can affect traffic flow. For example, the rules of network policies and security groups can affect traffic to the destination. Also, if multi-path is involved, then ECMP and RPF need to be taken into account while debugging.

Debugging BGP Peering and Route Exchange in Contrail

Use the troubleshooting steps and guidelines in this topic when you have errors with Contrail BGP peering and route exchange.

- [Example Cluster on page 636](#)
- [Verifying the BGP Routers on page 636](#)
- [Verifying the Route Exchange on page 639](#)
- [Debugging Route Exchange with Policies on page 641](#)
- [Debugging Peering with an MX Series Router on page 643](#)

- [Debugging a BGP Peer Down Error with Incorrect Family on page 645](#)
- [Configuring MX Peering \(iBGP\) on page 647](#)
- [Checking Route Exchange with an MX Series Peer on page 649](#)
- [Checking the Route in the MX Series Router on page 650](#)

Example Cluster

Examples in this document refer to a virtual cluster that is set up as follows:

Config Nodes : ['nodea22', 'nodea20']

Control Nodes : ['nodea22', 'nodea20']

Compute Nodes : ['nodea22', 'nodea20']

Collector : ['nodea22']

WebUI : nodea22

Openstack : nodea22

Verifying the BGP Routers

Use this procedure to launch various introspects to verify the setup of the BGP routers in your system.

Use this procedure to launch various introspects to verify the setup of the BGP routers in your system.

1. Verify the BGP routers.

All of the configured control nodes and external BGP routers are visible from the following location, shown using the sample node setup.

http://<host ip address>:8082/bgp-routers



NOTE: Throughout this procedure, replace <host ip address> with the correct location for your system to see the setup in your system.

Figure 269: Sample Output, BGP Routers:

```
{
  - bgp-routers: [
    - {
      href: "http://nodea22.englab.juniper.net:8082/bgp-router/1da579c5-0907-4c98-a7ad-37671f00cf60",
      - fq_name: [
        "default-domain",
        "default-project",
        "ip-fabric",
        "__default__",
        "nodea20"
      ],
      uuid: "1da579c5-0907-4c98-a7ad-37671f00cf60"
    },
    - {
      href: "http://nodea22.englab.juniper.net:8082/bgp-router/9702853f-5e48-417f-bd72-c00a12cc0200",
      - fq_name: [
        "default-domain",
        "default-project",
        "ip-fabric",
        "__default__",
        "nodea22"
      ],
      uuid: "9702853f-5e48-417f-bd72-c00a12cc0200"
    }
  ]
}
```

5041945

2. Verify the BGP peering.

The following statement is entered to check the **bgp_router_refs** object on the API server to validate the peering on the sample setup.

http://<host ip address>:8082/bgp-router/1da579c5-0907-4c98-a7ad-37671f00cf60

Figure 270: Sample Output, BGP Router References:

```

- bgp_router_parameters: {
  vendor: "contrail",
  autonomous_system: 64512,
  vnc_managed: null,
  address: "10.204.216.16",
  identifier: "10.204.216.16",
  port: 179,
  - address_families: {
    - family: {
      "inet-vpn",
      "e-vpn"
    }
  }
},
- bgp_router_refs: {
  - {
    - to: {
      "default-domain",
      "default-project",
      "ip-fabric",
      "___default___",
      "nodea22"
    },
    href: "http://nodea22.englab.juniper.net:8082/bgp-router/9702853f-5e48-417f-bd72-c00a12cc0200",
    - attr: {
      - session: {
        - attributes: {
          - {
            bgp_router: null,
            - address_families: {
              - family: {
                "inet-vpn",
                "e-vpn"
              }
            }
          },
          uuid: null
        }
      },
      uuid: "9702853f-5e48-417f-bd72-c00a12cc0200"
    }
  }
},
}

```

3. Verify the command line arguments that are passed to the control-node.

On the control-node, use **ps aux | grep control-node** to see the arguments that are passed to the control-node.

Example

```

/usr/bin/control-node --map-user <ip address> --map-password <ip
address>--hostname nodea22 --host-ip <ip address> --bgp-port 179
--discovery-server <ip address>

```

The hostname is the **bgp-router** name. Ensure that the bgp-router config can be found for the hostname, using the procedure in Step 1.

4. Validate the BGP neighbor config and the BGP peering config object.

http://<host ip address>:8083/Snh_ShowBgpNeighborConfigReq?

Figure 271: Sample Output, BGP Neighbor Config:

ShowBgpNeighborConfigResp						
neighbors						
instance_name	name	vendor	autonomous_system	identifier	address	address_families
default-domain:default-project:ip-fabric:___default___	default-domain:default-project:ip-fabric:___default___:nodea20	contrail	64512	10.204.216.16	10.204.216.16	address_families
						inet-vpn
						e-vpn

http://<host ip address>:8083/Snh_ShowBgpPeeringConfigReq?

Figure 272: Sample Output, BGP Peering Config:

ShowBgpPeeringConfigResp

peering		neighbor_count	session
instance_name	name		
default-domain:default-project:ip-fabric:...default...	addr(default-domain:default-project:ip-fabric:...default...mode20, default-domain:default-project:ip-fabric:...default...mode21)	1	sessions valid attributes attributes bgp_router address, families address, families inet-vpn e-vpn

5. Check the BGP neighbor states on the sample setup.

http://<host ip address>:8083/Snh_BgpNeighborReq?ip_address=&domain=

Figure 273: Sample Output, BGP Neighbor States:

BgpNeighborListResp

peer	peer_address	peer_asn	local_address	local_asn	encoding	peer_type	state	send_state	last_event	last_state	last_state_at	last_error
10.204.216.16	10.204.216.16	64512	10.204.216.18	64512	BGP	Internal	Established	In sync	Fac::EvBgpKeepalive	OpenConfirm	2014-Feb-10 07:08:21.177692	Unknown
10.204.216.253	10.204.216.253	64512	10.204.216.18	64512	BGP	Internal	Established	In sync	Fac::EvBgpUpdate	OpenConfirm	2014-Feb-10 11:24:45.851632	Cause:Administrator reset the peer

If the peer is not in an established state, check the **last_error** and the **flap_count**. Debug the BGP state machine by using information displayed in the output, such as **last_state** and **last_event**.



NOTE: The image displayed is truncated to fit this page. On the console screen you can scroll horizontally to see more columns and data.

Verifying the Route Exchange

The following two virtual networks are used in the sample debugging session for route exchange.

vn1 -> 1.1.1.0/24

vn2 -> 2.2.2.0/24

Example Procedure for Verifying Route Exchange

1. Validate the presence of the routing instance for each virtual network in the sample system.

http://<host ip address>:8083/Snh_ShowRoutingInstanceReq?name=



NOTE: Throughout this example, replace **<host ip address>** with the correct location for the control node on your system.

Figure 274: Sample Output, Show Routing Instance:

[illegible]

In the sample output, you can see the **import_target** and the **export_target** configured on the routing instance. Also shown are the **xmpp peers (vroutes)** registered to the table.

The user can click on the **inet** table of the required routing instance to display the routes that belong to the instance.

Use the information in Step 2 to validate a route.

2. Validate a route in a given routing instance in the sample setup:

http://<host ip>

```
address>:8083/Snh_ShowRouteReq?x=default-domain:demo:vn1:vn1.inet.0
```

In the following sample output (truncated), the user can validate the BGP paths for the protocol and for the source of the route to verify which XMPP agent or vRouter has pushed the route. If the path source is BGP, the route is imported to the VRF table from a BGP peer, either another control-node or an external bgp router such as an MX Series router. BGP paths are displayed in the order of path selection.

Figure 275: Sample Output, Validate Route:

ShowRouteMap						
tables						
routing_instance	routing_table_name	prefix	paths	primary_paths	secondary_paths	infeasible_paths
default-domain:demo:vni:vni1	default-domain:demo:vni:vni1.lnet.0	1	2	1	1	0
routes						
		prefix	last_modified		paths	
		1.1.1.255/32	2014-Feb-18 11:34:12.227000		paths protocol 300P BGP	

3. Validate the **l3vpn** table.

http://<host ip address>:8083/Snh_ShowRouteReq?x=bgp.l3vpn.0

ShowRouteResp

TABLE 1. (continued)

Category	Item	Score	Weight	Weighted Score	Subtotal	Grand Total
Knowledge	1. The following are the main components of a business plan: a) Executive Summary, b) Business Description, c) Market Analysis, d) Financial Projections, e) Management Team, f) Risk Analysis, g) Appendix. (10 marks)	10	10%	1.0	1.0	1.0
	2. The following are the main components of a business plan: a) Executive Summary, b) Business Description, c) Market Analysis, d) Financial Projections, e) Management Team, f) Risk Analysis, g) Appendix. (10 marks)	10	10%	1.0	2.0	2.0
	3. The following are the main components of a business plan: a) Executive Summary, b) Business Description, c) Market Analysis, d) Financial Projections, e) Management Team, f) Risk Analysis, g) Appendix. (10 marks)	10	10%	1.0	3.0	3.0
Skills	4. The following are the main components of a business plan: a) Executive Summary, b) Business Description, c) Market Analysis, d) Financial Projections, e) Management Team, f) Risk Analysis, g) Appendix. (10 marks)	10	10%	1.0	4.0	4.0
	5. The following are the main components of a business plan: a) Executive Summary, b) Business Description, c) Market Analysis, d) Financial Projections, e) Management Team, f) Risk Analysis, g) Appendix. (10 marks)	10	10%	1.0	5.0	5.0
	6. The following are the main components of a business plan: a) Executive Summary, b) Business Description, c) Market Analysis, d) Financial Projections, e) Management Team, f) Risk Analysis, g) Appendix. (10 marks)	10	10%	1.0	6.0	6.0
Attitudes	7. The following are the main components of a business plan: a) Executive Summary, b) Business Description, c) Market Analysis, d) Financial Projections, e) Management Team, f) Risk Analysis, g) Appendix. (10 marks)	10	10%	1.0	7.0	7.0
	8. The following are the main components of a business plan: a) Executive Summary, b) Business Description, c) Market Analysis, d) Financial Projections, e) Management Team, f) Risk Analysis, g) Appendix. (10 marks)	10	10%	1.0	8.0	8.0
	9. The following are the main components of a business plan: a) Executive Summary, b) Business Description, c) Market Analysis, d) Financial Projections, e) Management Team, f) Risk Analysis, g) Appendix. (10 marks)	10	10%	1.0	9.0	9.0
Total	10. The following are the main components of a business plan: a) Executive Summary, b) Business Description, c) Market Analysis, d) Financial Projections, e) Management Team, f) Risk Analysis, g) Appendix. (10 marks)	10	10%	1.0	10.0	10.0
	11. The following are the main components of a business plan: a) Executive Summary, b) Business Description, c) Market Analysis, d) Financial Projections, e) Management Team, f) Risk Analysis, g) Appendix. (10 marks)	10	10%	1.0	11.0	11.0
	12. The following are the main components of a business plan: a) Executive Summary, b) Business Description, c) Market Analysis, d) Financial Projections, e) Management Team, f) Risk Analysis, g) Appendix. (10 marks)	10	10%	1.0	12.0	12.0
Grand Total	13. The following are the main components of a business plan: a) Executive Summary, b) Business Description, c) Market Analysis, d) Financial Projections, e) Management Team, f) Risk Analysis, g) Appendix. (10 marks)	10	10%	1.0	13.0	13.0
	14. The following are the main components of a business plan: a) Executive Summary, b) Business Description, c) Market Analysis, d) Financial Projections, e) Management Team, f) Risk Analysis, g) Appendix. (10 marks)	10	10%	1.0	14.0	14.0
	15. The following are the main components of a business plan: a) Executive Summary, b) Business Description, c) Market Analysis, d) Financial Projections, e) Management Team, f) Risk Analysis, g) Appendix. (10 marks)	10	10%	1.0	15.0	15.0

Figure 278: Create Policy Window

Action	Protocol	Source Network	Source Ports	Direction	Destination Network	Destination Ports	Apply Service	Mirror to
PAS	ANY	default-domain:	Source	<>	default-domain:	Destinat	<input type="checkbox"/>	<input type="checkbox"/>

Buttons: Cancel, Create

2. Validate that the routing instances have the correct import_target configuration.

http://<host ip address>:8083/Snh_ShowRoutingInstanceReq?name=

Figure 279: Sample Output, Validate Import Target:

default-domain:demo:vn1:vn1	default-domain:demo:vn1	4	import_target target:64512:1 target:64512:2	export_target target:64512:1
default-domain:demo:vn2:vn2	default-domain:demo:vn2	5	import_target target:64512:1 target:64512:2	export_target target:64512:2

3. Validate that the routes are imported from VRF.

Use the BGP path attribute to check the replication status of the path. The route from the destination VRF should be replicated and validate the origin-vn.

Figure 280: Sample Output, Route Import:

ShowRouteResp

tables									
routing_instance	routing_table_name	prefixes	paths	primary_paths	secondary_paths	infeasible_paths	routes		
default-domain:demo:vn2:vn2	default-domain:demo:vn2:vn2:inet.0	2	4	1	3	0	prefix	last modified	paths
							1.1.1.253/32	2014-Feb-10 12:02:47.261344	paths
									protocol
									XMP
									BGP
							2.2.2.253/32	2014-Feb-10 11:34:35.469899	paths
									protocol
									XMP
									BGP

5041932

Debugging Peering with an MX Series Router

This section sets up an example BGP MX Series peer and provides some troubleshooting scenarios.

- 1. Set the Global AS number of the control-node for an MX Series BGP peer, using the Contrail WebUI (eBGP).

Figure 281: Edit Global ASN Window

Edit Global ASN

Global ASN

54321

5041932

Cancel

Save

- 2. Configure the eBGP peer for the MX Series router. Use the Contrail Web UI or Python provisioning.

Figure 282: Create BGP Peer Window

Configuring the MX Series BGP peer with the Python provision utility:

```
python ./provision_mx.py --router_name mx --router_ip <ip address>
--router_asn 12345 --api_server_ip <ip address> --api_server_port 8082
--oper add --admin_user admin --admin_password <password>
--admin_tenant_name admin
```

3. Configure a control-node peer on the MX Series router, using Junos CLI:

```
set protocols bgp group contrail-control-nodes type external
set protocols bgp group contrail-control-nodes local-address <ip address>
set protocols bgp group contrail-control-nodes keep all
set protocols bgp group contrail-control-nodes peer-as 54321
set protocols bgp group contrail-control-nodes local-as 12345
set protocols bgp group contrail-control-nodes neighbor <ip address>
```

Debugging a BGP Peer Down Error with Incorrect Family

Use this procedure to identify and resolve errors that arise from *families* mismatched configurations.



NOTE: This example uses locations at `http://<host ip address>:`. Be sure to replace `<host ip address>` with the correct address for your environment.

1. Check the BGP peer UVE.

`http://<host ip address>:8081/analytics/uves/bgp-peers`

2. Search for the MX Series BGP peer by name in the list.

In the sample output, **families** is the family advertised by the peer and **configured_families** is what is provisioned. In the sample output, the families configured on the peer has a mismatch, thus the peer doesn't move to an established state. You can verify it in the peer UVE.

Figure 283: Sample BGP Peer UVE

```
{
  - BgpPeerInfoData: {
    - state_info: {
      last_state: "Idle",
      state: "Idle",
      last_state_at: 1394778927107639
    },
    - families: [
      "IPv4:Unicast"
    ],
    peer_type: "external",
    local_asn: 54321,
    - configured_families: [
      "inet-vpn"
    ],
    - event_info: {
      last_event_at: 1394778927107880,
      last_event: "fsm::EvStart"
    },
    local_id: 181196816,
    send_state: "not advertising",
    peer_address: "10.204.216.253",
    peer_id: 181197053,
    hold_time: 90,
    peer_asn: 12345
  }
}
```

3. Fix the **families** mismatch in the sample by updating the configuration on the MX Series router, using Junos CLI:

`set protocols bgp group contrail-control-nodes family inet-vpn unicast`

4. After committing the CLI configuration, the peer comes up. Verify this with UVE.

`http://<host ip address>:8081/analytics/uves/bgp-peers`

Figure 284: Sample Established BGP Peer UVE

```
{
  - BgpPeerInfoData: {
    - state_info: {
      last_state: "OpenConfirm",
      state: "Established",
      last_state_at: 1394779652932460
    },
    - families: [
      "IPv4:Vpn"
    ],
    peer_type: "external",
    local_asn: 54321,
    - configured_families: [
      "inet-vpn"
    ],
    - event_info: {
      last_event_at: 1394779652992071,
      last_event: "fsm::EvBgpUpdate"
    },
    local_id: 181196816,
    send_state: "in sync",
    peer_address: "10.204.216.253",
    peer_id: 181197053,
    peer_asn: 12345
  }
}
```

5. Verify the peer status on the MX Series router, using Junos CLI:

```
run show bgp neighbor <ip address>
Peer: <ip address> AS 54321 Local: <ip address> AS 12345

Type: External    State: Established    Flags: <ImportEval Sync>

Last State: OpenConfirm    Last Event: RecvKeepAlive

Last Error: None

Options: <Preference LocalAddress KeepAll AddressFamily PeerAS LocalAS
Rib-group Refresh>

Address families configured: inet-vpn-unicast

Local Address: <ip address> Holdtime: 90 Preference: 170 Local AS: 12345
Local System AS: 64512

Number of flaps: 0

Error: 'Cease' Sent: 0 Recv: 2

Peer ID: <ip address>    Local ID: <ip address>    Active Holdtime: 90

Keepalive Interval: 30          Group index: 1    Peer index: 0

BFD: disabled, down

Local Interface: ge-1/0/2.0

NLRI for restart configured on peer: inet-vpn-unicast

NLRI advertised by peer: inet-vpn-unicast

NLRI for this session: inet-vpn-unicast
```

Peer does not support Refresh capability

Stale routes from peer are kept for: 300

Peer does not support Restarter functionality

Peer does not support Receiver functionality

Peer does not support 4 byte AS extension

Peer does not support Addpath

Configuring MX Peering (iBGP)

1. Edit the Global ASN.

Figure 285: Edit Global ASN Window

2. Configure the MX Series iBGP peer, using Contrail WebUI or Python provisioning.

Figure 286: Create BGP Peer Window

Configuring the MX Series BGP peer with the Python provision utility:

```
python ./provision_mx.py --router_name mx--router_ip <ip address> --router_asn 64512
--api_server_ip <ip address> --api_server_port 8082 --oper add --admin_user admin
--admin_password <password> --admin_tenant_name admin
```

3. Verify the peer from UVE.

<http://<host ip address>:8081/analytics/uves/bgp-peers>

Figure 287: Sample Established IBGP Peer UVE

```
{
  - BgpPeerInfoData: {
    - state_info: {
      last_state: "OpenConfirm",
      state: "Established",
      last_state_at: 1394788178225128
    },
    - families: {
      "IPv4:Vpn"
    },
    peer_type: "internal",
    local_asn: 64512,
    - configured_families: {
      "inet-vpn"
    },
    - event_info: {
      last_event_at: 1394788178267208,
      last_event: "fsm::EvBgpUpdate"
    },
    local_id: 181196816,
    send_state: "in_sync",
    peer_address: "10.204.216.253",
    peer_id: 181197053,
    peer_asn: 64512
  }
}
```

4. You can verify the same information at the HTTP introspect page of the control node (8443 in this example).

http://<host ip address>:8083/Snh_BgpNeighborReq?ip_address=&domain=

Figure 288: Sample Established IBGP Peer Introspect Window

neighbors											
peer	peer_address	peer_asn	local_address	local_asn	encoding	peer_type	state	send_state	last_event	last_state	
10.204.216.253	10.204.216.253	64512	10.204.216.16	64512	BGP	internal	Established	in_sync	fsm::EvBgpKeepalive	OpenConfirm	

Checking Route Exchange with an MX Series Peer

1. Check the route table in the bgp.l3vpn.0 table.

Figure 289: Routing Instance Route Table

routing_instance	routing_table_name	prefixes	paths	primary_paths	secondary_paths	infeasible_paths	routes
default-domain:default-project:ip-fabric:default...	bgp.l3vpn.0	2	2	2	0	0	routes prefix 10.204.216.253:5:0.0.0.0/0 10.204.216.253:5:10.204.218.0/24

2. Configure a public virtual network.

Figure 290: Routing Instance Route Table

routing_instance	routing_table_name	prefixes	paths	primary_paths	secondary_paths	infeasible_paths	routes
default-domain:default-project:ip-fabric:default...	bgp.l3vpn.0	2	2	2	0	0	routes prefix 10.204.216.253:5:0.0.0.0/0 10.204.216.253:5:10.204.218.0/24

3. Verify the routes in the public.inet.0 table.

http: //<host ip

address>:8083/Snh_ShowRouteReq?x=default-domain:admin:public:public.inet.0

Figure 291: Routing Instance Public IPv4 Route Table

tables	routing_instance	routing_table_name	prefixes	paths	primary_paths	secondary_paths	infeasible_paths	routes
	default-domain:admin:public:public	default-domain:admin:public:public.inet.0	2	2	0	2	0	routes prefix 0.0.0.0/0 10.204.218.0/24

4. Launch a virtual machine in the public network and verify the route in the public.inet.0 table.

http: //<host ip address>:8083/

Snh_ShowRouteReq?x=default-domain:admin:public:public.inet.0

Figure 292: Virtual Machine Routing Instance Public IPv4 Route Table

tables																																																
routing_instance	routing_table_name	prefixes	paths	primary_paths	secondary_paths	infeasible_paths	routes																																									
default-domain:admin:public:public	default-domain:admin:public:public.inet.0	3	3	1	2	0	<table><thead><tr><th>route</th><th>prefix</th><th>last_modified</th><th>paths</th><th>protocol</th></tr></thead><tbody><tr><td></td><td>0.0.0.0/0</td><td>2014-Mar-14 10:05:05.729526</td><td>paths</td><td>SCP</td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td>10.204.216.0/24</td><td>2014-Mar-14 10:05:05.729817</td><td>paths</td><td>SCP</td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td>11.2.3.253/32</td><td>2014-Mar-14 10:18:48.797958</td><td>paths</td><td>SCP</td></tr></tbody></table>	route	prefix	last_modified	paths	protocol		0.0.0.0/0	2014-Mar-14 10:05:05.729526	paths	SCP												10.204.216.0/24	2014-Mar-14 10:05:05.729817	paths	SCP												11.2.3.253/32	2014-Mar-14 10:18:48.797958	paths	SCP	0.0.0.0/0
route	prefix	last_modified	paths	protocol																																												
	0.0.0.0/0	2014-Mar-14 10:05:05.729526	paths	SCP																																												
	10.204.216.0/24	2014-Mar-14 10:05:05.729817	paths	SCP																																												
	11.2.3.253/32	2014-Mar-14 10:18:48.797958	paths	SCP																																												

5. Verify the route in the bgp.l3vpn.0 table.

http://<host ip address>:8083/Snh_ShowRouteReq?x=bgp.l3vpn.0

Figure 293: BGP Routing Instance Route Table

tables								
routing_instance	routing_table_name	prefixes	paths	primary_paths	secondary_paths	infeasible_paths	routes	
default-domain:default-project:ip-fabric:default...	bgp.l3vpn.0	3	3	2	1	0	<div>routes</div> <div>prefix</div> <div>10.204.216.253:5:0.0.0.0/0</div> <div>10.204.216.253:5:10.204.216.0/24</div> <div>10.204.216.70:1:11.2.3.253/32</div>	

Checking the Route in the MX Series Router

Use Junos CLI show commands from the router to check the route.

```
run show route table public.inet.0
```

```
public.inet.0: 5 destinations, 6 routes (5 active, 0 holddown, 0 hidden)
```

```
+ = Active Route, - = Last Active, * = Both
```

```
0.0.0.0/0          *[Static/5] 15w6d 08:50:34
```

```
> to <ip address> via ge-1/0/1.0
```

```
<ip address>     *[Direct/0] 15w6d 08:50:35
```

```
> via ge-1/0/1.0
```

```
<ip address>     *[Local/0] 15w6d 08:50:51
```

```
Local via ge-1/0/1.0
```

```

<ip address> *[BGP/170] 01:13:34, localpref 100, from <ip address>
    AS path: ?, validation-state: unverified
    > via gr-1/0/0.32771, Push 16
[BGP/170] 01:13:34, localpref 100, from <ip address>
    AS path: ?, validation-state: unverified
    > via gr-1/0/0.32771, Push 16

<ip address> *[BGP/170] 00:03:20, localpref 100, from <ip address>
    AS path: ?, validation-state: unverified
    > via gr-1/0/0.32769, Push 16

run show route table bgp.l3vpn.0 receive-protocol bgp <ip address> detail
bgp.l3vpn.0: 92 destinations, 130 routes (92 active, 0 holddown, 0 hidden)
* <ip address> (1 entry, 0 announced)

    Import Accepted

    Route Distinguisher: <ip address>

    VPN Label: 16

    Nexthop: <ip address>

    Localpref: 100

    AS path: ?

    Communities: target:64512:1 target:64512:10003 unknown iana 30c unknown
iana 30c unknown type 8004 value fc00:1 unknown type 8071 value fc00:4

```

Troubleshooting the Floating IP Address Pool in Contrail

This document provides troubleshooting methods to use when you have errors with the floating IP address pool when using Contrail.

- [Example Cluster on page 652](#)
- [Example on page 653](#)
- [Example: MX80 Configuration for the Gateway on page 654](#)
- [Ping the Floating IP from the Public Network on page 656](#)
- [Troubleshooting Details on page 656](#)
- [Get the UUID of the Virtual Network on page 656](#)
- [View the Floating IP Object in the API Server on page 657](#)

- [View floating-ips in floating-ip-pools in the API Server on page 660](#)
- [Check Floating IP Objects in the Virtual Machine Interface on page 662](#)
- [View the BGP Peer Status on the Control Node on page 665](#)
- [Querying Routes in the Public Virtual Network on page 666](#)
- [Verification from the MX80 Gateway on page 667](#)
- [Viewing the Compute Node Vnsw Agent on page 669](#)
- [Advanced Troubleshooting on page 672](#)

Example Cluster

Examples in this document refer to a virtual cluster that is set up as follows:

```
Config Nodes   : ['nodec6', 'nodec7', 'nodec8']
Control Nodes  : ['nodec7', 'nodec8']
Compute Nodes  : ['nodec9', 'nodec10']
Collector      : ['nodec6', 'nodec8']
WebUI          : nodec7
Openstack      : nodec6
```

The following virtual networks are used in the examples in this document:

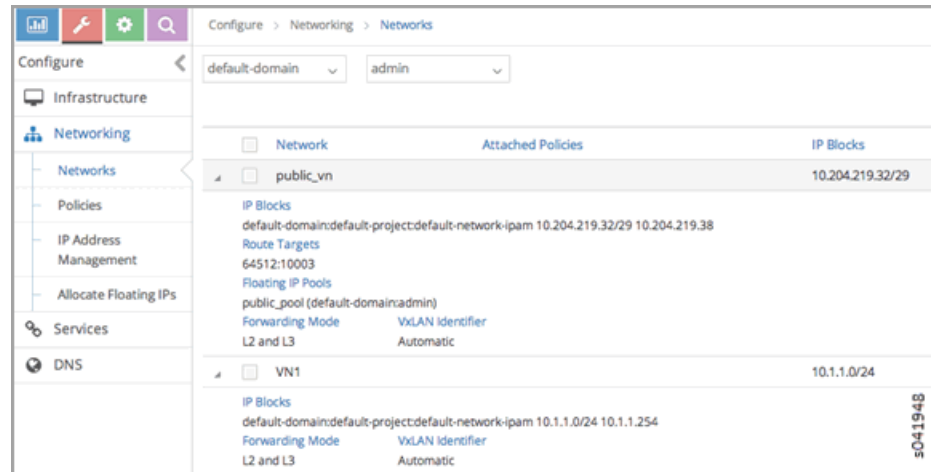
Public virtual network:

- Virtual network name: **public_vn**
- Public addresses range: **10.204.219.32 to 10.204.219.37**
- Route Target: **64512:10003**
- Floating IP pool name: **public_pool**

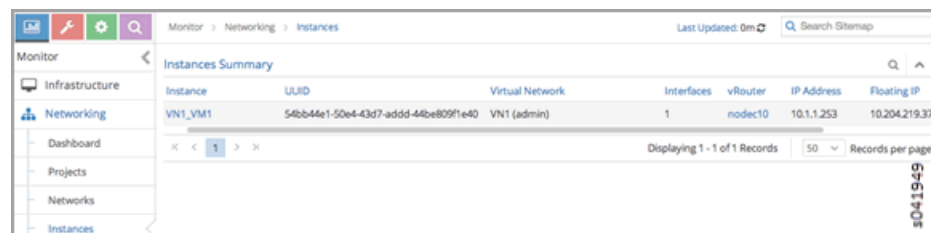
Private virtual network:

- Virtual network name: **vn1**
- Subnet: **10.1.1.0/24**

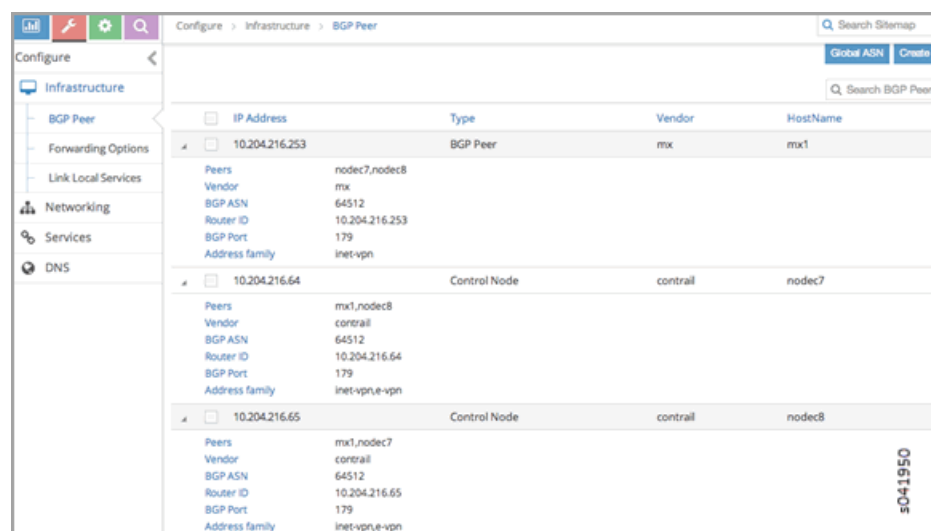
Example



A virtual machine is created in the virtual network VN1 with the name VN1_VM1 and with the IP address 10.1.1.253. A floating IP address of 10.204.219.37 is associated to the VN1_VM1 instance.



An MX80 router is configured as a gateway to peer with control nodes nodec7 and nodec8.



Example: MX80 Configuration for the Gateway

The following is the Junos OS configuration for the MX80 gateway. The route 10.204.218.254 is the route to the external world.

```
chassis {
  fpc 1 {
    pic 0 {
      tunnel-services;
    }
  }
}

interfaces {
  ge-1/0/1 {
    unit 0 {
      family inet {
        address 10.204.218.1/24;
      }
    }
  }
  ge-1/0/2 {
    unit 0 {
      family inet {
        address 10.204.216.253/24;
      }
    }
  }
}

routing-options {
  static {
    route 0.0.0.0/0 next-hop 10.204.216.254;
  }
  router-id 10.204.216.253;
}
```

```
route-distinguisher-id 10.204.216.253;

autonomous-system 64512;

dynamic-tunnels {
    tun1 {
        source-address 10.204.216.253;

        gre;

        destination-networks {
            10.204.216.0/24;
            10.204.217.0/24;
        }
    }
}

protocols {
    bgp {
        group control-nodes {
            type internal;

            local-address 10.204.216.253;

            keep all;

            family inet-vpn {
                unicast;
            }

            neighbor 10.204.216.64;

            neighbor 10.204.216.65;
        }
    }
}

routing-instances {
    public {
        instance-type vrf;
```

```
interface ge-1/0/1.0;

vrf-target target:64512:10003;

vrf-table-label;

routing-options {
    static {
        route 0.0.0.0/0 next-hop 10.204.218.254;
    }
}
}
```

Ping the Floating IP from the Public Network

From the public network, ping the floating IP 10.204.219.37.

```
user1-test:~ user1$ ping 10.204.219.37

PING 10.204.219.37 (10.204.219.37): 56 data bytes

64 bytes from 10.204.219.37: icmp_seq=0 ttl=54 time=62.439 ms
64 bytes from 10.204.219.37: icmp_seq=1 ttl=54 time=56.018 ms
64 bytes from 10.204.219.37: icmp_seq=2 ttl=54 time=55.915 ms
64 bytes from 10.204.219.37: icmp_seq=3 ttl=54 time=57.755 ms

^C
--- 10.204.219.37 ping statistics ---
5 packets transmitted, 4 packets received, 20.0% packet loss
round-trip min/avg/max/stddev = 55.915/58.032/62.439/2.647 ms
```

Troubleshooting Details

The following sections show details of ways to get related information, view, troubleshoot, and validate floating IP addresses in a Contrail system.

Get the UUID of the Virtual Network

Use the following to get the universal unique identifier (UUID) of the virtual network.

```
[root@nodec6 ~]# (source /etc/contrail/openstackrc; quantum net-list -F id -F name) 2>/dev/null
```

```
+-----+-----+
| id                | name                |
+-----+-----+
```



```

| 43707766-75f3-4d48-80d9-1b7240fb161d | public_vn |
| 2ab7ea04-8f5f-4b8d-acbf-a7c29c9b4112 | VN1 |
| 1c59ded0-38e8-4168-b91f-4c51aba10d30 | default-virtual-network |
| 5b0a1040-91e4-47ff-bd4c-0a81e1901a1f | ip-fabric |
| 7efddf64-ff3c-44d2-aeb2-45d7472b7a64 | __link_local__ |
+-----+-----+

```

View the Floating IP Object in the API Server

Use the following to view the floating IP pool information in the API server. API server requests can be made on http port 8082.

The Contrail API servers have the virtual-network public_vn object that contains floating IP pool information. Use the following to view the floating-ip-pools object information.

curl http://<API-Server_IP>:8082/virtual-network/<UUID_of_VN>

Example

```

root@nodec6 ~]# curl
http://nodec6:8082/virtual-network/43707766-75f3-4d48-80d9-1b7240fb161d |
python -m json.tool

{
  "virtual-network": {
    "floating_ip_pools": [
      {
        "href":
"http://127.0.0.1:8095/floating-ip-pool/663737c1-f3ab-40ff-9442-bdb6c225e3c3",

        "to": [
          "default-domain",
          "admin",
          "public_vn",
          "public_pool"
        ],
        "uuid": "663737c1-f3ab-40ff-9442-bdb6c225e3c3"
      }
    ],
  },
}

```

```
"fq_name": [
    "default-domain",
    "admin",
    "public_vn"
],
"href":
"http://127.0.0.1:8095/virtual-network/43707766-75f3-4d48-80d9-1b7240fb161d",

"id_perms": {
    "created": "2014-02-07T08:58:40.892803",
    "description": null,
    "enable": true,
    "last_modified": "2014-02-07T10:06:42.234423",
    "permissions": {
        "group": "admin",
        "group_access": 7,
        "other_access": 7,
        "owner": "admin",
        "owner_access": 7
    },
    "uuid": {
        "uuid_lslong": 9284482284331406877,
        "uuid_mslong": 4859515279882014024
    }
},
"name": "public_vn",
"network_ipam_refs": [
    {
        "attr": {
            "ipam_subnets": [
                {
                    "default_gateway": "10.204.219.38",
```

```

        "subnet": {
            "ip_prefix": "10.204.219.32",
            "ip_prefix_len": 29
        }
    },
    ],
    "href":
"http://127.0.0.1:8095/network-ipam/39b0e8da-fcd4-4b35-856c-8d18570b1483",
    "to": [
        "default-domain",
        "default-project",
        "default-network-ipam"
    ],
    "uuid": "39b0e8da-fcd4-4b35-856c-8d18570b1483"
}
],
    "parent_href":
"http://127.0.0.1:8095/project/deef6549-8e6c-4e3e-9cde-c9bc2b72ce6f",
    "parent_type": "project",
    "parent_uuid": "deef6549-8e6c-4e3e-9cde-c9bc2b72ce6f",
    "route_target_list": {
        "route_target": [
            "target:64512:10003"
        ]
    },
    },
    "routing_instances": [
        {
            "href":
"http://127.0.0.1:8095/routing-instance/3c6254ac-cfde-417e-916d-e7a1c0efad92",

            "to": [

```

```
        "default-domain",
        "admin",
        "public_vn",
        "public_vn"
    ],
    "uuid": "3c6254ac-cfde-417e-916d-e7a1c0efad92"
}
],
"uuid": "43707766-75f3-4d48-80d9-1b7240fb161d",
"virtual_network_properties": {
    "extend_to_external_routers": null,
    "forwarding_mode": "l2_l3",
    "network_id": 4,
    "vxlan_network_identifier": null
}
}
}
```

View floating-ips in floating-ip-pools in the API Server

Once you have located the floating-ip-pools object, use the following to review its floating-ips object.

The floating-ips object should display the floating IP that is shown in the Contrail UI. The floating IP should have a reference to the virtual machine interface (VMI) object that is bound to the floating IP.

Example

```
[root@nodec6 ~]#
curlhttp://nodec6:8082/floating-ip-pool/663737c1-f3ab-40ff-9442-bdb6c225e3c3
| python -m json.tool
```

```
{
  "floating-ip-pool": {
    "floating_ips": [
      {
```

```

        "href":
"http://127.0.0.1:8095/floating-ip/f3eec4d6-889e-46a3-a8f0-879dfaff6ca0",
        "to": [
            "default-domain",
            "admin",
            "public_vn",
            "public_pool",
            "f3eec4d6-889e-46a3-a8f0-879dfaff6ca0"
        ],
        "uuid": "f3eec4d6-889e-46a3-a8f0-879dfaff6ca0"
    }
],
    "fq_name": [
        "default-domain",
        "admin",
        "public_vn",
        "public_pool"
    ],
    "href":
"http://127.0.0.1:8095/floating-ip-pool/663737c1-f3ab-40ff-9442-bdb6c225e3c3",

    "id_perms": {
        "created": "2014-02-07T08:58:41.136572",
        "description": null,
        "enable": true,
        "last_modified": "2014-02-07T08:58:41.136572",
        "permissions": {
            "group": "admin",
            "group_access": 7,
            "other_access": 7,
            "owner": "admin",
            "owner_access": 7
        }
    }
}

```

```
    },
    "uuid": {
        "uuid_lslong": 10683309858715198403,
        "uuid_mslong": 7365417021744038143
    }
},
"name": "public_pool",
"parent_href":
"http://127.0.0.1:8095/virtual-network/43707766-75f3-4d48-80d9-1b7240fb161d",

"parent_type": "virtual-network",
"parent_uuid": "43707766-75f3-4d48-80d9-1b7240fb161d",
"project_back_refs": [
    {
        "attr": {},
        "href":
"http://127.0.0.1:8095/project/deef6549-8e6c-4e3e-9cde-c9bc2b72ce6f",
        "to": [
            "default-domain",
            "admin"
        ],
        "uuid": "deef6549-8e6c-4e3e-9cde-c9bc2b72ce6f"
    }
],
"uuid": "663737c1-f3ab-40ff-9442-bdb6c225e3c3"
}
```

Check Floating IP Objects in the Virtual Machine Interface

Use the following to retrieve the virtual machine interface of the virtual machine from either the quantum port-list command or from the Contrail UI. Then get the virtual machine interface identifier and check its floating IP object associations.

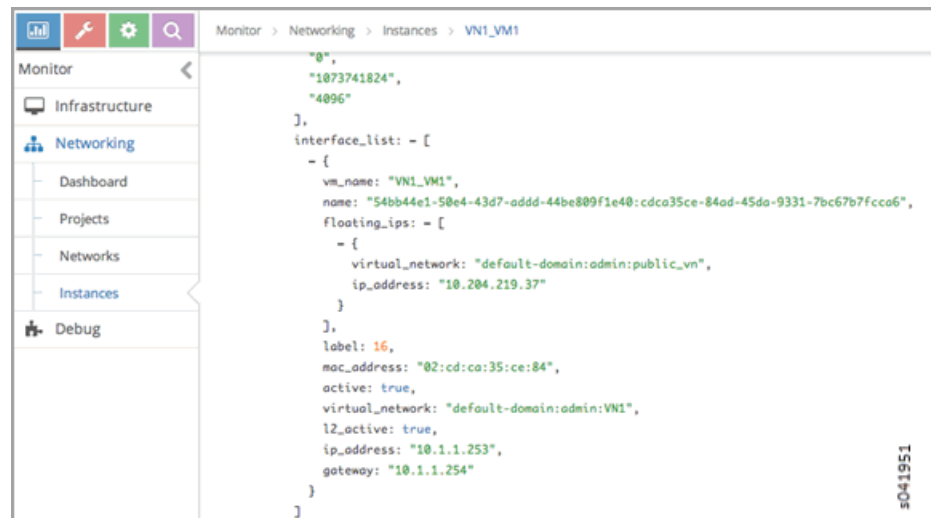
- Using **quantum port-list** to get the virtual machine interface:

Example

```
[root@nodec6 ~]# quantum port-list -F id -F fixed_ips
```

id		fixed_ips	
cdca35ce-84ad-45da-9331-7bc67b7fcc66		{ "subnet_id": "e80f480b-98d4-43cc-847c-711e637295db", "ip_address": "10.1.1.253" }	

- Using Contrail UI to get the virtual machine interface:



Checking Floating IP Objects on the Virtual Machine Interface

Once you have obtained the virtual machine interface identifier, check the floating-ip objects that are associated with the virtual machine interface.

```
[root@nodec6 ~]# curl
http://127.0.0.1:8095/floating-ip/f3eec4d6-889e-46a3-a8f0-879dfaff6ca0 | python
-m json.tool
```

```
{
  "floating-ip": {
    "floating_ip_address": "10.204.219.37",
    "fq_name": [
```

```
        "default-domain",
        "admin",
        "public_vn",
        "public_pool",
        "f3eec4d6-889e-46a3-a8f0-879dfaff6ca0"
    ],
    "href":
    "http://127.0.0.1:8095/floating-ip/f3eec4d6-889e-46a3-a8f0-879dfaff6ca0",
    "id_perms": {
        "created": "2014-02-07T10:07:05.869899",
        "description": null,
        "enable": true,
        "last_modified": "2014-02-07T10:36:36.820926",
        "permissions": {
            "group": "admin",
            "group_access": 7,
            "other_access": 7,
            "owner": "admin",
            "owner_access": 7
        },
        "uuid": {
            "uuid_lslong": 12173378905373109408,
            "uuid_mslong": 17577202821367744163
        }
    },
    "name": "f3eec4d6-889e-46a3-a8f0-879dfaff6ca0",
    "parent_href":
    "http://127.0.0.1:8095/floating-ip-pool/663737c1-f3ab-40ff-9442-bdb6c225e3c3",
    "parent_type": "floating-ip-pool",
    "parent_uuid": "663737c1-f3ab-40ff-9442-bdb6c225e3c3",
    "project_refs": [
```



```

    {
      "attr": null,
      "href":
"http://127.0.0.1:8095/project/deef6549-8e6c-4e3e-9cde-c9bc2b72ce6f",
      "to": [
        "default-domain",
        "admin"
      ],
      "uuid": "deef6549-8e6c-4e3e-9cde-c9bc2b72ce6f"
    }
  ],
  "uuid": "f3eec4d6-889e-46a3-a8f0-879dfaff6ca0",
  "virtual_machine_interface_refs": [
    {
      "attr": null,
      "href":
"http://127.0.0.1:8095/virtual-machine-interface/cdca35ce-84ad-45da-9331-7bc67b7fcca6",

      "to": [
        "54bb44e1-50e4-43d7-addd-44be809f1e40",
        "cdca35ce-84ad-45da-9331-7bc67b7fcca6"
      ],
      "uuid": "cdca35ce-84ad-45da-9331-7bc67b7fcca6"
    }
  ]
}

```

View the BGP Peer Status on the Control Node

Use the Contrail UI or the control node http introspect on port 8083 to view the BGP peer status. In the following example, the control nodes are **nodec7** and **nodec8**.

Ensure that the BGP peering state is displayed as **Established** for the control nodes and the gateway MX.

Example

- Using the Contrail UI:

The screenshot shows the Contrail UI interface. The left sidebar contains navigation links: Monitor, Infrastructure, Control Nodes, Virtual Routers, Analytics Nodes, Config Nodes, and Networking. The main panel is titled 'Monitor > Infrastructure > Control Nodes > nodec8'. It has tabs for Details, Peers, Routes, and Console. The 'Peers' tab is active, displaying a table with the following data:

Peer	Peer Type	Peer ASN	Status	Last Rap	Messages (Recv/Sent)
10.204.216.253	BGP	64512	Established, in sync	-	1707/ 1590
10.204.216.64	BGP	64512	Established, in sync	2/7/2014 11:46:32 AM	1595/ 1597

At the bottom of the table, it says 'Displaying 1 - 2 of 2 Records' and '50 Records per page'. The bottom right corner shows 's041952'.

- Using the control-node Introspect:

`http://nodec7:8083/Snh_BgpNeighborReq?ip_address=&domain=`

`http://nodec8:8083/Snh_BgpNeighborReq?ip_address=&domain=`

Querying Routes in the Public Virtual Network

On each control-node, a query on the routes in the **public_vn** lists the routes that are pushed by the MX gateway, which in the following example are 0.0.0.0/0 and 10.204.218.0/24.

In the following results, the floating IP route of 10.204.217.32 is installed by the compute node (nodec10) that hosts that virtual machine.

Example

- Using the Contrail UI:

The screenshot shows the Contrail UI interface. The left sidebar is the same as the previous screenshot. The main panel is titled 'Monitor > Infrastructure > Control Nodes > nodec8'. It has tabs for Details, Peers, Routes, and Console. The 'Routes' tab is active, displaying a form and a table. The form has fields for 'Routing Instance' (set to 'default-domainadminpublic_vnpublic_vn'), 'Address Family' (set to 'All'), and 'Limit' (set to '50 Routes'). There are also fields for 'Peer Source', 'Prefix', and 'Protocol'. Below the form is a 'Display Routes' button. The 'Routes' table below shows the following data:

Routing Table	Prefix	Protocol	Source	Next hop	Label	Secur...	Origin VN
default-domainadminpublic_vnpublic_vn.inet.0	0.0.0.0/0	BGP	10.204.216.253	10.204.216.253	16	-	default-domainadminpublic_vn
	10.204.218.0/24	BGP	10.204.216.253	10.204.216.253	16	-	default-domainadminpublic_vn
	10.204.217.32/32	XMPP	nodec10	10.204.216.67	16	1	default-domainadminpublic_vn

- Using the http Introspect:

Following is the format for using an introspect query.

`http://<nodename/ip>:8083/Snh_ShowRouteReq?x=<RoutingInstance of public VN>.inet.0`

Example

http://nodec8:8083/Snh_BgpNeighborReq?ip_address=&domain=

routing instance	routing table name	prefix	path	primary path	secondary path	insecure path	routes
default-domain-admin-public-vm-public-vm	default-domain-admin-public-vm-public-vm	8.8.8.8/8	10.204.216.64	10.204.216.64	10.204.216.64	10.204.216.64	BGP
default-domain-admin-public-vm-public-vm	default-domain-admin-public-vm-public-vm	10.204.216.64/24	10.204.216.64	10.204.216.64	10.204.216.64	10.204.216.64	BGP
default-domain-admin-public-vm-public-vm	default-domain-admin-public-vm-public-vm	10.204.216.37/32	10.204.216.64	10.204.216.64	10.204.216.64	10.204.216.64	BGP

View Corresponding BGP LL3VPN Routes

Use the Contrail UI or the http introspect to view the public route's corresponding BGP LL3VPN routes, as in the following.

Example

- Using the Contrail UI:

Routing Table	Prefix	Protocol	Source	Next Hop	Label	Security	Origin
bgp.ll3vpn.0	10.204.216.253/16	BGP	10.204.216.253	10.204.216.253	16	-	-
	10.204.216.253/16	BGP	10.204.216.253	10.204.216.253	16	-	-
	10.204.216.67/32	XMPP	nodec10	10.204.216.67	16	1	default-domain-admin/vn
		BGP	10.204.216.64	10.204.216.67	16	1	default-domain-admin/vn
	10.204.216.67/32	XMPP	nodec10	10.204.216.67	16	1	default-domain-admin/vn
		BGP	10.204.216.64	10.204.216.67	16	1	default-domain-admin/vn

- Using the control-node Introspect:

http://nodec7:8083/Snh_ShowRouteReq?x=bgp.ll3vpn.0

http://nodec8:8083/Snh_ShowRouteReq?x=bgp.ll3vpn.0

Verification from the MX80 Gateway

This section provides options for verifying floating IP pools from the MX80 gateway.

Verify BGP Sessions are Established

Use the following commands from the gateway to verify that BGP sessions are established with the control nodes nodec7 and nodec8:

```
root@mx-host> show bgp neighbor 10.204.216.64
```

```
Peer: 10.204.216.64+59287 AS 64512 Local: 10.204.216.253+179 AS 64512
```

```
Type: Internal      State: Established      Flags: <Sync>
Last State: OpenConfirm  Last Event: RecvKeepAlive
Last Error: Hold Timer Expired Error
Options: <Preference LocalAddress KeepAll AddressFamily Rib-group Refresh>
Address families configured: inet-vpn-unicast
Local Address: 10.204.216.253 Holdtime: 90 Preference: 170
Number of flaps: 216
Last flap event: HoldTime
Error: 'Hold Timer Expired Error' Sent: 68 Recv: 0
Error: 'Cease' Sent: 0 Recv: 43
Peer ID: 10.204.216.64  Local ID: 10.204.216.253  Active Holdtime: 90
Keepalive Interval: 30      Group index: 0      Peer index: 3
BFD: disabled, down
NLRI for restart configured on peer: inet-vpn-unicast
NLRI advertised by peer: inet-vpn-unicast
NLRI for this session: inet-vpn-unicast
Peer does not support Refresh capability
Stale routes from peer are kept for: 300
Peer does not support Restarter functionality
Peer does not support Receiver functionality
Peer does not support 4 byte AS extension
Peer does not support Addpath
```

**Show Routes Learned
from Control Nodes**

From the MX80, use show route to display the routes for the virtual machine 10.204.219.37 that are learned from both control-nodes.

In the following example, the routes learned are 10.204.216.64 and 10.204.216.65, pointing to a dynamic GRE tunnel next hop with a label of 16 (of the virtual machine).

```
public.inet.0: 4 destinations, 5 routes (4 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

0.0.0.0/0          *[Static/5] 10w6d 18:47:50
```

```
> to 10.204.218.254 via ge-1/0/1.0
10.204.218.0/24 *[Direct/0] 10w6d 18:47:51
> via ge-1/0/1.0
10.204.218.1/32 *[Local/0] 10w6d 18:48:07
Local via ge-1/0/1.0
10.204.219.37/32 *[BGP/170] 09:42:43, localpref 100, from 10.204.216.64
AS path: ?, validation-state: unverified
> via gr-1/0/0.32779, Push 16
[BGP/170] 09:42:43, localpref 100, from 10.204.216.65
AS path: ?, validation-state: unverified
> via gr-1/0/0.32779, Push 16
```

Viewing the Compute Node Vnsw Agent

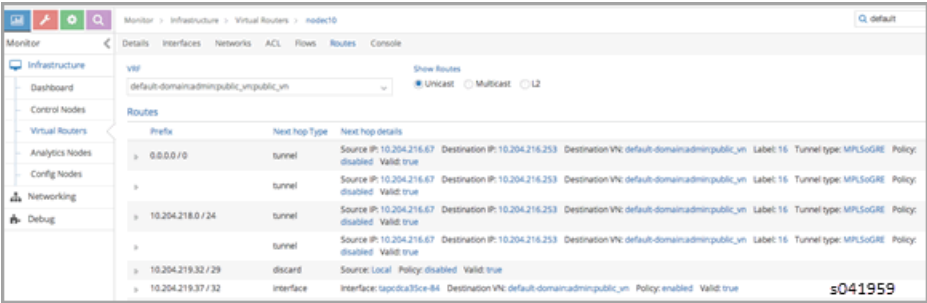
The compute node introspect can be accessed from port 8085. In the following examples, the compute nodes are nodec9 and nodec10.

View Routing Instance
Next Hops

On the routing instance of VN1, the routes 0.0.0.0/0 and 10.204.218.0/24 should have the next hop pointing to the MX gateway (10.204.216.253).

Example

Using the Contrail UI:



Prefix	Next hop Type	Next hop details
0.0.0.0/0	tunnel	Source IP: 10.204.216.67 disabled Valid: true Destination IP: 10.204.216.253 Destination VNI: default-domainadminpublic_vn Label: 16 Tunnel type: MPLSoGRE Policy: disabled
10.204.218.0/24	tunnel	Source IP: 10.204.216.67 disabled Valid: true Destination IP: 10.204.216.253 Destination VNI: default-domainadminpublic_vn Label: 16 Tunnel type: MPLSoGRE Policy: disabled
10.204.219.32/29	discard	Source: Local Policy: disabled Valid: true
10.204.219.37/32	interface	Interface: tap0ca35ce-84 Destination VNI: default-domainadminpublic_vn Policy: enabled Valid: true

Using the Unicast
Route Table Index to
View Next Hops

Alternatively, from the agent introspect, you can view the next hops at the unicast route table.

First, use the following to get the unicast route table index (ucindex) for the routing instance **default-domain:admin:public_vn:public_vn**.

http://nodec10:8085/Snh_VrfListReq?x=default-domain:admin:public_vn:public_vn

Example

In the following example, the unicast route table index is 2.

name	ucindex	mindex	l2index
default-domain:admin:public_vn:public_vn	2	2	2

Next, perform a route request query on ucindex 2, as shown in the following. The tunnel detail indicates the source and destination endpoints of the tunnel and the MPLS label 16 (the label of the virtual machine).

The query should also show a route for 10.204.219.37 with an interface next hop of tap-interface. **http://nodec10:8085/Snh_Inet4UcRouteReq?x=2**

src_ip	src_port	src_vrf	path_info	label	vrfm_id	peer
10.204.216.67	0	default-domain:admin:public_vn:public_vn	type: tunnel ref_count: 4 valid: true policy: disabled ip: 10.204.216.67 dip: 10.204.216.253 vrf: default-domain:default-project:ip-fabric:...default... mac: 8:81:F4:B4:7e:52 tunnel_type: vni_SuGRE	16	0	10.204.216.64
10.204.216.67	0	default-domain:admin:public_vn:public_vn	type: tunnel ref_count: 4 valid: true policy: disabled ip: 10.204.216.67 dip: 10.204.216.253 vrf: default-domain:default-project:ip-fabric:...default... mac: 8:81:F4:B4:7e:52 tunnel_type: vni_SuGRE	16	0	10.204.216.65

10.204.219.37	32	default-domain:admin:public_vn:public_vn	path_list			
			nh	label	vrf_id	peer
			nh	16	0	10.204.216.64
			type	interface		
			ref_count	g		
			valid	true		
			policy	enabled		
			if	tapcdca35ce-84		
			mac	2:cd:ce:35:ce:84		
			mcast	disabled		
			nh	16	0	10.204.216.65
			type	interface		
			ref_count	g		
			valid	true		
			policy	enabled		
			if	tapcdca35ce-84		
			mac	2:cd:ce:35:ce:84		
			mcast	disabled		

A ping from the MX gateway to the virtual machine's floating IP in the public routing-instance should work.

Advanced Troubleshooting

If you still have reachability problems after performing all of the tests in this article, for example, a ping between the virtual machine and the MX IP or to public addresses is failing, try the following:

- Validate that all the required Contrail processes are running by using the **contrail-status** command on all of the nodes.
- On the compute node where the virtual machine is present (nodec10 in this example), perform a tcpdump on the tap interface (**tcpdump -ni tapcdca35ce-84**). The output should show the incoming packets from the virtual machine.

- Check to see if any packet drops occur in the kernel vrouter module:

http://nodec10:8085/Snh_KDropStatsReq?

In the output, scroll down to find any drops. Note: You can ignore any ds_invalid_arp increments.

- On the physical interface where packets transmit onto the compute-node, perform a tcpdump matching the host IP of the MX to show the GRE encapsulated packets, as in the following.

```
[root@nodec10 ~]# cat /etc/contrail/agent.conf |grep -A 1 eth-port
```

```
<eth-port>
```

```
<name>p1p0p0</name>
```

```
</eth-port>
```

```
<metadata-proxy>
```

```
[root@nodec10 ~]# tcpdump -ni p1p0p0 host 10.204.216.253 -vv
```



```
tcpdump: WARNING: p1p0p0: no IPv4 address assigned

tcpdump: listening on p1p0p0, link-type EN10MB (Ethernet), capture size
65535 bytes

02:06:51.729941 IP (tos 0x0, ttl 64, id 57430, offset 0, flags [DF], proto
GRE (47), length 112)

    10.204.216.253 > 10.204.216.67: GREv0, Flags [none], length 92

        MPLS (label 16, exp 0, [S], ttl 54)

            IP (tos 0x0, ttl 54, id 35986, offset 0, flags [none], proto ICMP (1),
length 84)

                172.29.227.6 > 10.204.219.37: ICMP echo request, id 53240, seq 242,
length 64

02:06:51.730052 IP (tos 0x0, ttl 64, id 324, offset 0, flags [none], proto
GRE (47), length 112)

    10.204.216.67 > 10.204.216.253: GREv0, Flags [none], length 92

        MPLS (label 16, exp 0, [S], ttl 64)

            IP (tos 0x0, ttl 64, id 33909, offset 0, flags [none], proto ICMP (1),
length 84)

                10.204.219.37 > 172.29.227.6: ICMP echo reply, id 53240, seq 242, length
64

02:06:52.732283 IP (tos 0x0, ttl 64, id 12675, offset 0, flags [DF], proto
GRE (47), length 112)

    10.204.216.253 > 10.204.216.67: GREv0, Flags [none], length 92

        MPLS (label 16, exp 0, [S], ttl 54)

            IP (tos 0x0, ttl 54, id 54155, offset 0, flags [none], proto ICMP (1),
length 84)

                172.29.227.6 > 10.204.219.37: ICMP echo request, id 53240, seq 243,
length 64

02:06:52.732355 IP (tos 0x0, ttl 64, id 325, offset 0, flags [none], proto
GRE (47), length 112)

    10.204.216.67 > 10.204.216.253: GREv0, Flags [none], length 92

        MPLS (label 16, exp 0, [S], ttl 64)

            IP (tos 0x0, ttl 64, id 33910, offset 0, flags [none], proto ICMP (1),
length 84)

                10.204.219.37 > 172.29.227.6: ICMP echo reply, id 53240, seq 243, length
64

^C

4 packets captured
```

```
5 packets received by filter
0 packets dropped by kernel
[root@nodec10 ~]#
```

- On the MX gateway, use the following to inspect the GRE tunnel rx/tx (received/transmitted) packet count:

```
root@mx-host> show interfaces gr-1/0/0.32779 |grep packets

Input packets : 542

Output packets: 559
```

```
root@blr-mx1> show interfaces gr-1/0/0.32779 |grep packets

Input packets : 544

Output packets: 561
```

- Look for any packet drops in the FPC, as in the following:

```
show pfe statistics traffic fpc <id>
```

- Also inspect the dynamic tunnels, using the following:

```
show dynamic-tunnels database
```

Removing Stale Virtual Machines and Virtual Machine Interfaces

This topic gives examples for removing stale VMs (virtual machines) and VMIs (virtual machine interfaces). Before you can remove a stale VM or VMI, you must first remove any back references associated to the VM or VMI.

- [Problem Example on page 674](#)
- [Show Virtual Machines on page 675](#)
- [Show Virtual Machines Using Python API on page 677](#)
- [Delete Methods on page 678](#)

Problem Example

The troubleshooting examples in this topic are based on the following problem example. A **net-delete** of the virtual machine 2a8120ec-bd18-49f4-aca0-acfc6e8fe74f returned the following messages that there are two VMIs that still have back-references to the stale VM.

The two VMIs must be deleted first, then the Neutron **net-delete <vm_ID>** command will complete without errors.

```
From neutron.log:
```

```
2014-03-10 14:18:05.208
```

```

DEBUG [urllib3.connectionpool]
"DELETE/virtual-network/2a8120ec-bd18-49f4-aca0-acfc6e8fe74f HTTP/1.1" 409
203
2014-03-10 14:18:05.278
ERROR [neutron.api.v2.resource] delete failed
Traceback (most recent call last):
  File "/usr/lib/python2.7/dist-packages/neutron/api/v2/resource.py", line
84, in resource
    result = method(request=request, **args)
  File "/usr/lib/python2.7/dist-packages/neutron/api/v2/base.py", line
432, in delete
    obj_deleter(request.context, id, **kwargs)
  File
"/usr/lib/python2.7/dist-packages/neutron/plugins/juniper/contrail/contrail
plugin.py", line 294, in delete_network
    raise e
RefsExistError: Back-References from
http: //127.0.0.1:8082/virtual-machine-interface/51daf6f4-7366-4463-a819-bd1
17fe3a8c8,
http: //127.0.0.1:8082/virtual-machine-interface/30882e66-e175-4fbb-862e-354
bb700b579 still exist

```

Show Virtual Machines

Use the following command to show all of the virtual machines known to the Contrail API server. Replace the variable **<config-node-IP>** shown in the example with the IP address of the **config-node** in your setup.

http://<config-node-IP>:8082/virtual-machines

Example

In the following example, 03443891-99cc-4784-89bb-9d1e045f8aa6 is a stale VM that needs to be removed.

virtual-machines:

```

[
{

```

```
href:"http:
//example-node:8082/virtual-machine/03443891-99cc-4784-89bb-9d1e045f8aa6",

fq_name:

[

"03443891-99cc-4784-89bb-9d1e045f8aa6"

],

uuid:"03443891-99cc-4784-89bb-9d1e045f8aa6"

},
```

When the user attempts to delete the stale VM, a message displays that children to the VM still exist:

```
root@example-node:~# curl -X DELETE -H "Content-Type: application/json;
charset=UTF-8" http:
//127.0.0.1:8082/virtual-machine/03443891-99cc-4784-89bb-9d1e045f8aa6
Children http:
//127.0.0.1:8082/virtual-machine-interface/0c32a82a-7bd3-46c7-b262-6d85b9911a0d
still exist
root@example-node:~#
```

The user opens `http://example-node:8082/virtual-machine/03443891-99cc-4784-89bb-9d1e045f8aa6`, and sees a **virtual-machine-interface** (VMI) attached to it. The VMI must be removed before the VM can be removed.

However, when the user attempts to delete the VMI from the stale VM, they get a message that there is still a back-reference:

```
root@example-node:~# curl -X DELETE -H "Content-Type: application/json;
charset=UTF-8" http:
//<example-IP>:8082/virtual-machine-interface/0c32a82a-7bd3-46c7-b262-6d85b9911a0d

Back-References from http:
//<example-IP>:8082/instance-ip/6ffa29a1-023f-462b-b205-353da8e3a2a4 still
exist

root@example-node:~#
```

Because there is a back-reference from an **instance-ip** object still present, the **instance-ip** object must first be deleted, as follows:

```
root@example-node:~# curl -X DELETE -H "Content-Type: application/json;
charset=UTF-8" http:
//<example-IP>:8082/instance-ip/6ffa29a1-023f-462b-b205-353da8e3a2a4

root@example-node:~#
```

When the **instance-ip** is deleted, then the VMI and the VM can be deleted.



NOTE: To prevent inconsistency, be certain that the VM is not present in the Nova database before deleting the VM.

Show Virtual Machines Using Python API

The following example shows how to view virtual machines using a Python API. This example shows virtual machines and back-references. Once you identify back-references and existing children, you can delete them first, then delete the stale VM.

```
root@example-node:~# source /opt/contrail/api-venv/bin/activate

File "<stdin>", line 1, in <module>

  File
"/opt/contrail/api-venv/lib/python2.7/site-packages/vnc_api/gen/vnc_api_client_gen.py",
  line 3793, in virtual_machine_interface_delete

    content = self._request_server(rest.OP_DELETE, uri)

  File "/opt/contrail/api-venv/lib/python2.7/site-packages/vnc_api/vnc_api.py",
  line 342, in _request_server

    raise RefsExistError(content)

cfgm_common.exceptions.RefsExistError: Back-References from http: //
<example-IP>:8082/instance-ip/6ffa29a1-023f-462b-b205-353da8e3a2a4 still exist

>>> (api-venv)root@example-node:~# python

Python 2.7.5 (default, Mar 10 2014, 03:55:35)

[GCC 4.6.3] on linux2

Type "help", "copyright", "credits" or "license" for more information.

>>> from vnc_api.vnc_api import VncApi

>>> vh=VncApi()

>>>
vh.virtual_machine_interface_delete(id='0c32a82a-7bd3-46c7-b262-6d85b9911a0d')

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

  File
"/opt/contrail/api-venv/lib/python2.7/site-packages/vnc_api/gen/vnc_api_client_gen.py",
  line 3793, in virtual_machine_interface_delete

    content = self._request_server(rest.OP_DELETE, uri)

  File "/opt/contrail/api-venv/lib/python2.7/site-packages/vnc_api/vnc_api.py",
  line 342, in _request_server

    raise RefsExistError(content)

cfgm_common.exceptions.RefsExistError: Back-References from http: //
<example-IP>:8082/instance-ip/6ffa29a1-023f-462b-b205-353da8e3a2a4 still exist

>>>
```

Delete Methods

Use help (**vh**) to show all delete methods supported.

Typical commands for deleting VMs and VMIs include:

- **virtual_machine_delete()** to delete a virtual machine
- **instance_ip_delete()** to delete an **instance-ip**.

Troubleshooting Link-Local Services in Contrail

Use the troubleshooting steps and guidelines in this topic when you have errors with Contrail link-local services.

- [Overview of Link-Local Services on page 678](#)
- [Troubleshooting Procedure for Link-Local Services on page 678](#)
- [Metadata Service on page 681](#)
- [Troubleshooting Procedure for Link-Local Metadata Service on page 681](#)

Overview of Link-Local Services

Virtual machines might be set up to access specific services hosted on the fabric infrastructure. For example, a virtual machine might be a Nova client that requires access to the Nova API service running in the fabric network. Access to services hosted on the fabric network can be provided by configuring the services as link-local services.

A link-local address and a service port is chosen for the specific service running on a TCP / UDP port on a server in the fabric. With the link-local service configured, virtual machines can access the service using the link-local address. For link-local services, Contrail uses the address range 169.254.169.x.

Link-local service can be configured using the Contrail WebUI: **Configure > Infrastructure > Link Local Services**.

Service Name	Link Local Service Address		Fabric Address	
	IP	Port	IP / DNS	Port
ntp	169.254.169.100	123	IP 172.17.28.5	123

Troubleshooting Procedure for Link-Local Services

Use the following steps when you are troubleshooting link-local services errors.

1. Verify the reachability of the fabric server that is hosting the link-local service from the compute node.
2. Check the state of the virtual machine and the interface:
 - Is the **Status** of virtual machine **Up**?
 - Is the corresponding tap interface **Active**?

Checking the virtual machine status in the Contrail UI:

Monitor > Infrastructure > Virtual Routers > nodec15							Search Sitemap
Details	Interfaces	Networks	ACL	Flows	Routes	Console	
Interfaces							
Name	Label	Status	Network	IP Address	Floating IP	Instance	
tap4b094dbe-f0	18	Up	vn1 (demo)	1.2.3.247	None	4f4b917a-a071-4517-961a-0e41067fec53 / vn1-vm2	

Checking the tap interface status in the http agent introspect:

http://<compute-node-ip>:8085/Snh_ItfReq?name=

itf_list						
index	name	uuid	vrf_name	active		
3	tap722a7a11-6d	722a7a11-6d2e-47e9-a4cc-687a105a240f	default-domain:demo:vn1:vn1	Active		s041995

3. Check the link-local configuration in the vrouter agent. Make sure the configured link-local service is displayed.

http://<compute-node-ip>:8085/Snh_LinkLocalServiceInfo?

service_list						
linklocal_service_name	linklocal_service_ip	linklocal_service_port	ipfabric_dns_name	ipfabric_ip	ipfabric_port	
ntp	169.254.169.100	123	-	ipfabric_ip 172.17.28.5	123	s041996

4. Validate the BGP neighbor config and the BGP peering config object. When the virtual machine communicates with the configured link-local service, a forward and reverse flow for the communication is set up. Check that the flow for this communication is created and the flow action is NAT.

http://<compute-node-ip>:8085/Snh_KFlowReq?flow_idx=

Check that all flow entries display NAT action programmed and display flags for the fields (source or destination IP and ports) that have NAT programmed. Also shown are the number of packets and bytes transmitted in the respective flows.

flow_list									
index	rflow	sip	sport	dip	dport	proto	vrf_id	action	flags
467472	234436	1.2.3.247	123	169.254.169.100	123	17	1	NAT	ACTIVE VRFT SNAT SPAT DNAT SPAT
234436	467472	172.17.28.5	123	10.284.216.72	43226	17	0	NAT	ACTIVE VRFT SNAT DNAT s041997

The forward flow displays the source IP of the virtual machine and the destination IP of the link-local service. The reverse flow displays the source IP of the fabric host and the destination IP of the compute node's vhost interface. If the service is hosted on

the same compute node, the destination address of the reverse flow displays the metadata address allocated to the virtual machine.

Note that the **index** and **rflow** index for the two flows are reversed.

You can also view similar information in the vrouter agent introspect page, where you can see the policy and security group for the flow. Check that the flow actions display as **pass**.

http://<compute-node-ip>:8085/Snh_FetchAllFlowRecords?

Metadata Service

OpenStack allows virtual instances to access metadata by sending an HTTP request to the link-local address 169.254.169.254. The metadata request from the instance is proxied to Nova, with additional HTTP header fields added, which Nova uses to identify the source instance. Then Nova responds with appropriate metadata.

The Contrail router acts as the proxy, trapping the metadata requests, adding the necessary header fields, and sending the requests to the Nova API server.

Troubleshooting Procedure for Link-Local Metadata Service

Metadata service is also a link-local service, with a fixed service name (metadata), a fixed service address (169.254.169.254:80), and a fabric address pointing to the server where the OpenStack Nova API server is running. All of the configuration and troubleshooting procedures for Contrail link-local services also apply to the metadata service.

However, for metadata service, the flow is always set up to the compute node, so the router agent will update and proxy the HTTP request. The router agent listens on a local port to receive the metadata requests. Consequently, the reverse flow has the compute node as the source IP, the local port on which the agent is listening is the source port, and the instance's metadata IP is the destination IP address.

After performing all of the troubleshooting procedures for link-local services, the following additional steps can be used to further troubleshoot metadata service.

1. Check the metadata statistics for: the number of metadata requests received by the router agent, the number of proxy sessions set up with the Nova API server, and number of internal errors encountered.

`http://<compute-node-ip>:8085/Snh_MetadataInfo?`

The port on which the router agent listens for metadata requests is also displayed.

metadata_server_port	45094
metadata_requests	2
metadata_responses	0
metadata_proxy_sessions	2
metadata_internal_errors	0

2. Check the metadata trace messages, which show the trail of metadata requests and

responses.

`http://<compute-node-ip>:8085/Snh_SandeshTraceRequest?x=Metadata`

3. Check the Nova configuration. On the server running the OpenStack service, inspect the **nova.conf** file.

- Ensure that the metadata proxy is enabled, as follows:

`service_neutron_metadata_proxy = True`

`service_quantum_metadata_proxy = True` (on older installations)

- Check to see if the metadata proxy shared secret is set:

`neutron_metadata_proxy_shared_secret`

`quantum_metadata_proxy_shared_secret` (on older installations)

If the shared secret is set in **nova.conf**, the same secret must be configured on each compute node in the file **/etc/contrail/contrail-vrouter-agent.conf**, and the same shared secret must be updated in the **METADATA** section as

`metadata_proxy_secret=<secret>`.

4. Restart the vrouter agent after modifying the shared secret:

`service contrail-vrouter restart`

PART 5

Contrail Commands and APIs

- [Contrail Commands on page 685](#)
- [Contrail Application Programming Interfaces \(APIs\) on page 709](#)

CHAPTER 27

Contrail Commands

- [Getting Contrail Node Status on page 685](#)
- [contrail-logs \(Accessing Log File Messages\) on page 694](#)
- [contrail-status \(Viewing Node Status\)](#)
- [contrail-version \(Viewing Version Information\)](#)
- [service \(Managing Services\)](#)
- [Backing Up Contrail Databases Using JSON Format on page 702](#)

Getting Contrail Node Status

- [Overview on page 685](#)
- [UVE for NodeStatus on page 685](#)
- [Node Status Features on page 686](#)
- [Using Introspect to Get Process Status on page 691](#)
- [contrail-status script on page 692](#)

Overview

This topic describes how to view the status of a Contrail node on a physical server. Contrail nodes include config, control, analytics, compute, and so on.

UVE for NodeStatus

The User-Visible Entity (UVE) mechanism is used to aggregate and send the status information. All node types send a NodeStatus structure in their respective node UVEs. The following is a control node UVE of NodeStatus:

```
struct NodeStatus {  
    1: string name (key="ObjectBgpRouter")  
    2: optional bool deleted  
    3: optional string status  
    // Sent by process  
    4: optional list<process_info.ProcessStatus> process_status  
    (aggtype="union")
```

```
// Sent by node manager

5: optional list<process_info.ProcessInfo> process_info (aggtype="union")

6: optional string description
}

uve sandesh NodeStatusUVE {
    1: NodeStatus data
}
```

Node Status Features

The most important features of NodeStatus include:

ProcessStatus

ProcessInfo

ProcessStatus Also process_status, is sent by the processes corresponding to the virtual node, and displays the status of the process and an aggregate state indicating if the process is functional or non-functional. The process_status includes the state of the process connections (ConnectionInfo) to important services and other information necessary for the process to be functional. Each process sends its NodeStatus information, which is aggregated as union (aggtype="union") at the analytics node. The following is the ProcessStatus structure:

```
1. struct ProcessStatus {
2.     1: string module_id
3.     2: string instance_id
4.     3: string state
5.     4: optional list<ConnectionInfo> connection_infos
6.     5: optional string description
7. }
8.
9. struct ConnectionInfo {
10.    1: string type
11.    2: string name
12.    3: optional list<string> server_addrs
13.    4: string status
```

```

14.      5: optional string description
15.  }

```

ProcessInfo Sent by the node manager, /usr/bin/contrail-nodemgr. Node manager is a monitor process per contrail virtual node that tracks the running state of the processes. The following is the ProcessInfo structure:

```

16. struct ProcessInfo {
17.     1: string                process_name
18.     2: string                process_state
19.     3: u32                   start_count
20.     4: u32                   stop_count
21.     5: u32                   exit_count
22.     // time when the process last entered running stage
23.     6: optional string       last_start_time
24.     7: optional string       last_stop_time
25.     8: optional string       last_exit_time
26.     9: optional list<string> core_file_list
27. }

```

Example: NodeStatus The following is an example output of NodeStatus obtained from the Rest API:

```

http://:8081/analytics/uves/control-...ilt=NodeStatus .
{
  NodeStatus:
  {
    process_info:
    [
      {
        process_name: "contrail-control",
        process_state: "PROCESS_STATE_RUNNING",
        last_stop_time: null,
        start_count: 1,
        core_file_list: [ ],
        last_start_time: "1409002143776558",
        stop_count: 0,

```

```
    last_exit_time: null,
    exit_count: 0
  },
{
  process_name: "contrail-control-nodemgr",
  process_state: "PROCESS_STATE_RUNNING",
  last_stop_time: null,
  start_count: 1,
  core_file_list: [ ],
  last_start_time: "1409002141773481",
  stop_count: 0,
  last_exit_time: null,
  exit_count: 0
},
{
  process_name: "contrail-dns",
  process_state: "PROCESS_STATE_RUNNING",
  last_stop_time: null,
  start_count: 1,
  core_file_list: [ ],
  last_start_time: "1409002145778383",
  stop_count: 0,
  last_exit_time: null,
  exit_count: 0
},
{
  process_name: "contrail-named",
  process_state: "PROCESS_STATE_RUNNING",
  last_stop_time: null,
```



```

    start_count: 1,
    core_file_list: [ ],
    last_start_time: "1409002147780118",
    stop_count: 0,
    last_exit_time: null,
    exit_count: 0
  }
],
process_status:
[
{
  instance_id: "0",
  module_id: "ControlNode",
  state: "Functional",
  description: null,
  connection_infos:
  [
    {
      server_addrs:
      [
        "10.84.13.45:8443"
      ],
    }
  ],
  server_addrs:
  [
    "10.84.13.45:8086"
  ],
  status: "Up",
  type: "Collector",
  name: null,
  description: "Established"
}
]

```

```
    },  
  
    {  
      server_addrs:  
      [  
        "10.84.13.45:5998"  
      ],  
      status: "Up",  
      type: "Discovery",  
      name: "Collector",  
      description: "SubscribeResponse"  
    },  
  
    {  
      server_addrs:  
      [  
        "10.84.13.45:5998"  
      ],  
      status: "Up",  
      type: "Discovery",  
      name: "IfmapServer",  
      description: "SubscribeResponse"  
    },  
  
    {  
      server_addrs:  
      [  
        "10.84.13.45:5998"  
      ],  
      status: "Up",  
      type: "Discovery",  
      name: "xmpp-server",  
      description: "Publish Response - HeartBeat"
```

```

    }
  ]
}
]
}
}

```

Using Introspect to Get Process Status

The user can also view the state of a specific process by using the introspect mechanism.

Example: Introspect of NodeStatus

The following is an example of the process state of contrail-control that is obtained by using

`http://server-ip:8083/Snh_SandeshUVECacheReq?x=NodeStatus`



NOTE: The example output is the ProcessStatus of only one process of contrail-control. It does not show the full aggregated status of the control node through its UVE (as in the previous example).

```
root@a6s45:~# curl http://10.84.13.45:8083/Snh_SandeshU...q?x=NodeStatus
```

```

<?xml-stylesheet type="text/xsl"
href="/universal_parse.xsl"?><__NodeStatusUVE_list type="slist"><NodeStatusUVE
  type="sandesh"><data type="struct" identifier="1"><NodeStatus><name
type="string" identifier="1" key="ObjectBgpRouter">a6s45</name><process_status
  type="list" identifier="4" aggttype="union"><list type="struct"
size="1"><ProcessStatus><module_id type="string"
identifier="1">ControlNode</module_id><instance_id type="string"
identifier="2">0</instance_id><state type="string"
identifier="3">Functional</state><connection_infos type="list"
identifier="4"><list type="struct" size="5"><ConnectionInfo><type type="string"
  identifier="1">IFMap</type><name type="string"
identifier="2">IFMapServer</name><server_addrs type="list" identifier="3"><list
  type="string"
size="1"><element>10.84.13.45:8443</element></list></server_addrs><status
type="string" identifier="4">Up</status><description type="string"
identifier="5">Connection with IFMap Server
(irond)</description></ConnectionInfo><ConnectionInfo><type type="string"
identifier="1">Collector</type><name type="string"
identifier="2"></name><server_addrs type="list" identifier="3"><list
type="string"
size="1"><element>10.84.13.45:8086</element></list></server_addrs><status
type="string" identifier="4">Up</status><description type="string"
identifier="5">Established</description></ConnectionInfo><ConnectionInfo><type
type="string" identifier="1">Discovery</type><name type="string"
identifier="2">Collector</name><server_addrs type="list" identifier="3"><list
type="string"
size="1"><element>10.84.13.45:5998</element></list></server_addrs><status
type="string" identifier="4">Up</status><description type="string"
identifier="5">SubscribeResponse</description></ConnectionInfo><ConnectionInfo><type
type="string" identifier="1">Discovery</type><name type="string"

```

```

    identifier="2">IfmapServer</name><server_addrs type="list" identifier="3"><list
      type="string"
      size="1"><element>10.84.13.45:5998</element></list></server_addrs><status
      type="string" identifier="4">Up</status><description type="string"
      identifier="5">SubscribeResponse</description></ConnectionInfo><ConnectionInfo<type
      type="string" identifier="1">Discovery</type><name type="string"
      identifier="2">xmpp-server</name><server_addrs type="list" identifier="3"><list
      type="string"
      size="1"><element>10.84.13.45:5998</element></list></server_addrs><status
      type="string" identifier="4">Up</status><description type="string"
      identifier="5">Publish Response -
      HeartBeat</description></ConnectionInfo></list></connection_infos><description
      type="string"
      identifier="5"></description></ProcessStatus></list></process_status><NodeStatus><data><NodeStatusUVE><SandeshUVECacheResp
      type="sandesh"><returned type="u32" identifier="1">1</returned><more
      type="bool"
      identifier="0">false</more></SandeshUVECacheResp></__NodeStatusUVE_list>

```

contrail-status script

The contrail-status script is used to give the status of the Contrail processes on a server.

The contrail-status script first checks if a process is running, and if it is, performs introspect into the process to get its functionality status, then outputs the aggregate status.

The possible states to display include:

- active - the process is running and functional; the internal state is good
- inactive - not started or stopped by user
- failed – the process exited too quickly and has not restarted
- initializing - the process is running, but the internal state is not yet functional.

Example Output: Contrail-Status Script

The following is an example output from the contrail-status script.

```

root@a6s45:~# contrail-status

== Contrail vRouter ==

supervisor-vrouter:           active

contrail-vrouter-agent        active

contrail-vrouter-nodemgr      active


== Contrail Control ==

supervisor-control:          active

contrail-control              active

contrail-control-nodemgr     active

contrail-dns                  active

```

```
contrail-named          active
```

```
== Contrail Analytics ==
```

```
supervisor-analytics:   active
```

```
contrail-analytics-api   active
```

```
contrail-analytics-nodemgr active
```

```
contrail-collector       active
```

```
contrail-query-engine    active
```

```
== Contrail Config ==
```

```
supervisor-config:      active
```

```
contrail-api:0          active
```

```
contrail-config-nodemgr active
```

```
contrail-schema         active
```

```
contrail-svc-monitor    active
```

```
rabbitmq-server         active
```

```
== Contrail Web UI ==
```

```
supervisor-webui:       active
```

```
contrail-webui          active
```

```
contrail-webui-middleware active
```

```
redis-webui             active
```

```
== Contrail Database ==
```

```
supervisord-contrail-database:active
```

```
contrail-database       active
```

```
contrail-database-nodemgr active
```

contrail-logs (Accessing Log File Messages)

A command-line utility, **contrail-logs**, uses REST APIs to retrieve system log messages, object log messages, and trace messages.

- [Command-Line Options for Contrail-Logs on page 694](#)
- [Option Descriptions on page 694](#)
- [Example Uses on page 695](#)

Command-Line Options for Contrail-Logs

The command-line utility for accessing log file information is **contrail-logs** in the analytics node. The following are the options supported at the command line for **contrail-logs**, as viewed using the **--help** option.

```
[root@host]# contrail-logs --help
usage: contrail-logs [-h]
                    [--opserver-ip OPSERVER_IP]
                    [--opserver-port OPSERVER_PORT]
                    [--start-time START_TIME]
                    [--end-time END_TIME]
                    [--last LAST]
                    [--source SOURCE]
                    [--module {ControlNode, VRouterAgent, ApiServer, Schema,
OpServer, Collector, QueryEngine, ServiceMonitor, DnsAgent}]
                    [--category CATEGORY]
                    [--level LEVEL]
                    [--message-type MESSAGE_TYPE]
                    [--reverse]
                    [--verbose]
                    [--all]
                    [--object {ObjectVNTTable, ObjectVMTable, ObjectSITable,
ObjectVRouter, ObjectBgpPeer, ObjectRoutingInstance, ObjectBgpRouter,
ObjectXmppConnection, ObjectCollectorInfo, ObjectGeneratorInfo,
ObjectConfigNode}]
                    [--object-id OBJECT_ID]
                    [--object-select-field {ObjectLog, SystemLog}]
                    [--trace TRACE]
```

Option Descriptions

The following are the descriptions for each of the option arguments available for **contrail-logs**.

```
optional arguments:
  -h, --help                show this help message and exit
  --opserver-ip OPSERVER_IP
                           IP address of OpServer (default: 127.0.0.1)
  --opserver-port OPSERVER_PORT
                           Port of OpServer (default: 8081)
  --start-time START_TIME
                           Logs start time (format now-10m, now-1h) (default:
now-10m)
  --end-time END_TIME
```

```

--last LAST           Logs end time (default: now)
--source SOURCE       Logs from last time period (format 10m, 1d) (default:
None)
--source SOURCE       Logs from source address (default: None)
--module {ControlNode, VRouterAgent, ApiServer, Schema, OpServer, Collector,
QueryEngine, ServiceMonitor, DnsAgent}
--category CATEGORY  Logs from module (default: None)
--category CATEGORY  Logs of category (default: None)
--level LEVEL         Logs of level (default: None)
--message-type MESSAGE_TYPE
--message-type MESSAGE_TYPE  Logs of message type (default: None)
--reverse            Show logs in reverse chronological order (default:
False)
--verbose            Show internal information (default: True)
--all                Show all logs (default: False)
--object {ObjectVNTTable, ObjectVMTable, ObjectSITable, ObjectVRouter,
ObjectBgpPeer, ObjectRoutingInstance, ObjectBgpRouter, ObjectXmppConnection,
ObjectCollectorInfo, ObjectGeneratorInfo, ObjectConfigNode}
--object-id OBJECT_ID  Logs of object type (default: None)
--object-id OBJECT_ID  Logs of object name (default: None)
--object-select-field {ObjectLog, SystemLog}
--object-select-field {ObjectLog, SystemLog}  Select field to filter the log (default: None)
--trace TRACE        Dump trace buffer (default: None)

```

Example Uses

The following examples show how you can use the option arguments available for **contrail-logs** to retrieve the information you specify.

1. View only the system log messages from all boxes for the last 10 minutes.

contrail-logs

2. View all log messages (systemlog, objectlog, uve, ...) from all boxes for the last 10 minutes.

contrail-logs --all

3. View only the control node system log messages from all boxes for the last 10 minutes.

contrail-logs --module ControlNode

--module accepts the following values - **ControlNode, VRouterAgent, ApiServer, Schema, ServiceMonitor, Collector, OpServer, QueryEngine, DnsAgent**

4. View the control node system log messages from source **a6s23.contrail.juniper.net** for the last 10 minutes.

contrail-logs --module ControlNode --source a6s23.contrail.juniper.net

5. View the XMPP category system log messages from all modules on all boxes for the last 10 minutes.

contrail-logs --category XMPP

6. View the system log messages from all the boxes from the last hour.

contrail-logs --last 1h

7. View the system log messages from the VN object named **demo:admin:vn1** from all boxes for the last 10 minutes.

contrail-logs --object ObjectVNTTable --object-id demo:admin:vn1

--object accepts the following values - **ObjectVNTTable, ObjectVMTable, ObjectSITable, ObjectVRouter, ObjectBgpPeer, ObjectRoutingInstance, ObjectBgpRouter, ObjectXmppConnection, ObjectCollectorInfo**

8. View the system log messages from all boxes for the last 10 minutes in reverse chronological order:

contrail-logs --reverse

9. View the system log messages from a specific time interval and display them in a specified date format.

contrail-logs --start-time "2013 May 12 18:30:27.0" --end-time "2013 May 12 18:31:27.0"

contrail-status (Viewing Node Status)

Syntax	<code>[root@host ~]# contrail-status</code>
Release Information	Command introduced in Contrail Release 1.0.
Description	Display a list of all components of a Contrail server node (such as control, configuration, database, Web-UI, analytics, or vrouter) and report their current status of active or inactive.
Required Privilege Level	admin

Sample Output

The following example usage displays on a server that is configured for the roles of **vrouter**, **controller**, **analytics**, **configuration**, **web-ui**, and **database**.

Sample Output

```

root@host:~# contrail-status
== Contrail vRouter ==
supervisor-vrouter:           active
contrail-vrouter-agent        active
contrail-vrouter-nodemgr      active

== Contrail Control ==
supervisor-control:           active
contrail-control              active
contrail-control-nodemgr      active
contrail-dns                  active
contrail-named                active

== Contrail Analytics ==
supervisor-analytics:         active
contrail-analytics-api        active
contrail-analytics-nodemgr    active
contrail-collector            active
contrail-query-engine         active

== Contrail Config ==
supervisor-config:            active
contrail-api:0                active
contrail-config-nodemgr       active
contrail-discovery:0          active
contrail-schema               active
contrail-svc-monitor          active
ifmap                         active
rabbitmq-server               active

== Contrail Web UI ==
supervisor-webui:             active
contrail-webui                active
contrail-webui-middleware     active
redis-webui                   active

== Contrail Database ==

```

```
supervisord-contrail-database:active
contrail-database                active
contrail-database-nodemgr       active
```

contrail-version (Viewing Version Information)

Syntax [root@host]# contrail-version

Release Information Command introduced in Contrail Release 1.0.

Description Display a list of all installed components with their version and build numbers.

Required Privilege Level admin

Sample Output

The following example shows version and build information for all installed components.

Sample Output

```
root@host> contrail-version
Package                               Version                               Build-ID | Repo |
RPM Name
-----
contrail-analytics                    1-1309090026.e16                    141
contrail-analytics-venv               0.1-1309062310.e16                  141
contrail-api                          0.1-1309090026.e16                  141
contrail-api-lib                      0.1-1309090026.e16                  141
contrail-api-venv                     0.1-1309080539.e16                  141
contrail-control                      2012.0-1309090026.e16               141
contrail-database                     0.1-1309050028                      141
contrail-dns                          1-1309090026.e16                    141
contrail-fabric-utils                 1-1309090026                        141
contrail-libs                         1-1309090026.e16                    141
contrail-nodejs                       0.8.15-1309090026.e16               141
contrail-openstack-analytics          0.1-1309090026.e16                  141
contrail-openstack-cfgm               0.1-1309090026.e16                  141
contrail-openstack-control            0.1-1309090026.e16                  141
```

Sample Output

The following example shows version and build information for only the installed contrail components.

Sample Output

```
root@host> contrail-version | grep contrail
Package                               Version                               Build-ID | Repo |
RPM Name
-----
contrail-analytics                    1-1309090026.e16                    141
contrail-analytics-venv               0.1-1309062310.e16                  141
contrail-api                          0.1-1309090026.e16                  141
```

contrail-api-lib	0.1-1309090026.e16	141
contrail-api-venv	0.1-1309080539.e16	141
contrail-control	2012.0-1309090026.e16	141
contrail-database	0.1-1309050028	141
contrail-dns	1-1309090026.e16	141
contrail-fabric-utils	1-1309090026	141
contrail-libs	1-1309090026.e16	141
contrail-nodejs	0.8.15-1309090026.e16	141
contrail-openstack-analytics	0.1-1309090026.e16	141
contrail-openstack-cfgm	0.1-1309090026.e16	141
contrail-openstack-control	0.1-1309090026.e16	141
contrail-openstack-database	0.1-1309090026.e16	141
contrail-openstack-webui	0.1-1309090026.e16	141
contrail-setup	1-1309090026.e16	141
contrail-webui	1-1309090026	141
openstack-quantum-contrail	2013.2-1309090026	141

service (Managing Services)

Syntax	<code>service contrail-service (start stop restart status)</code>
Release Information	Standard Linux command used for managing and viewing services in Contrail Controller Release 1.0.
Description	<p>Start, stop, or restart a Contrail service. Display the status of a Contrail service.</p> <p>All contrail services are managed by the process supervisord, which is open source software written in Python. Each Contrail node type, such as compute, control, and so on, has an instance of supervisord that, when running, launches Contrail services as child processes. All supervisord instances display in contrail-status output with the prefix supervisor. If the supervisord instance of a particular node type is not up, none of the services for that node type are up. For more details about the open source supervisord process, see http://www.supervisord.org.</p>
Options	<ul style="list-style-type: none"> • start—start a named service. • stop—stop a named service. • restart—stop and restart a named service. • status—display the status of a named service.
Required Privilege Level	admin

Sample Output

The following examples show usage for the **contrail-collector** service, which is only configured on nodes that have the roles of **analytics**, **configuration**, **web-ui**, or **database**.

Sample Output

```
[root@host service supervisor-analytics status
supervisord (pid 32116) is running... [
[root@host]# service contrail-collector restart

contrail-collector: stopped
contrail-collector: started
[root@host]# service contrail-collector stop

contrail-collector: stopped
[root@host]# service contrail-collector start

contrail-collector: started
[root@host]# service contrail-collector status

contrail-collector          RUNNING    pid 20071, uptime 0:00:04
```

Backing Up Contrail Databases Using JSON Format

This document shows how to backup Contrail databases (Cassandra and Zookeeper) using a JSON format. Instructions are given for both non-containerized and containerized versions of Contrail, starting with Contrail 4.0.

- [Preliminary Cautions on page 702](#)
- [Simple Backup Using JSON Format on page 702](#)
- [Restore Simple Database Backup on page 703](#)
- [Example Backup and Restore With JSON on page 704](#)

Preliminary Cautions



CAUTION: Because the state of the Contrail database is associated with other system databases, such as OpenStack databases, database backups must be consistent across all systems and database changes associated with northbound APIs must be stopped on all systems before performing any backup operation. For example, you might block the external VIP for northbound APIs at the load balancer level, such as HAProxy.

Simple Backup Using JSON Format

Perform a simple backup (database dump). Working from a controller node, use `db_json_exim.py`, located at `/usr/lib/python2.7/dist-packages/cfgm_common`.



NOTE: The controller node for non-containerized Contrail is a virtual machine (VM).

The controller node for containerized Contrail is a controller container.

```
cd /usr/lib/python2.7/dist-packages/cfgm_common
```

```
python db_json_exim.py --export-to db-dump.json
```

- To see a cleaner version of the dump.

```
cat db-dump.json | python -m json.tool | less
```

- To omit keyspace in the dump, for example, to share with Juniper.

```
python db_json_exim.py --export-to db-dump.json --omit-keyspace dm_keyspace
```

Restore Simple Database Backup

Use the following steps to restore a system from a simple backup.

1. Stop **supervisor-config** on all controllers, if present, or ensure it is already stopped.

```
service supervisor-config stop
```

2. Stop Cassandra on all **config-db** controllers or ensure it is already stopped.

```
service cassandra stop
```

3. Stop Zookeeper on all controllers or ensure it is already stopped.

```
service zookeeper stop
```

4. Stop Kafka on all controllers. Be sure to check analytics controllers.

```
service kafka stop
```

5. Stop **contrail-hamon** on all controllers, if it is running on controllers.

```
service contrail-hamon stop
```

6. Backup the Zookeeper data directory on all controllers.

```
cd /var/lib/zookeeper/
```

```
cp -R version-2/ version-2-save
```

7. Backup the Cassandra data directory on all controllers.

```
cd /var/lib/
```

```
cp -R cassandra cassandra-save
```

8. Wipe out the Zookeeper data directory contents on all controllers.

```
rm -rf /var/lib/zookeeper/version-2/*
```

9. Wipe out the Cassandra data directory contents on all controllers.

```
rm -rf /var/lib/cassandra/*
```

10. Start Zookeeper on all controllers.

```
service zookeeper start
```

11. Start Cassandra on all controllers.

```
service cassandra start
```

12. Run `python db_json_exim.py --import-from db-dump.json` on any one controller.

```
cd /usr/lib/python2.7/dist-packages/cfgm_common
python db_json_exim.py --import-from db-dump.json
```

13. Start `supervisor-config` on all controllers (if present).

```
service supervisor-config start
```

14. Start Kafka on all controllers (check in analytics controllers).

```
service kafka start
```

15. Start `contrail-hamon` on all controllers, if previously stopped.

```
service contrail-hamon start
```

Example Backup and Restore With JSON

This section provides an example of a simple database backup and restore of a system that has three controllers with config-db and separate IPs with the following host IDs:

- 5b5s42
- 5b5s43
- 5b5s44

Example: Perform Simple Backup

```
root@5b5s42:~# python db_json_exim.py --export-to db-dump.json
root@5b5s42:~# cat db-dump.json | python -m json.tool | less
{
  "cassandra": {
    "config_db_uuid": {
      "objfq_name_table": {
        "access_control_list": {

<snip>
```


Example: Perform Restore

1. Stop **supervisor-config** on all controllers, if present.

Non-Containerized Version:

```

root@5b5s42:~# service supervisor-config stop
supervisor-config stop/waiting
root@5b5s42:~#
root@5b5s43:~# service supervisor-config stop
supervisor-config stop/waiting
root@5b5s43:~#
root@5b5s44:~# service supervisor-config stop
supervisor-config stop/waiting
root@5b5s44:~#

```

Containerized Version:

```

root@host-4.1:~# docker ps
CONTAINER ID        IMAGE                                     COMMAND                  CREATED             STATUS              PORTS
8802395bc033        172.30.109.59:5100/contrail410-contrail-analytics:mainline
"/lib/systemd/syst... 7 weeks ago         Up 2 weeks
analytics
f5aed0a2efc3        172.30.109.59:5100/contrail410-contrail-analyticsdb:mainline
"/lib/systemd/syst... 7 weeks ago         Up 2 weeks
analyticsdb
0ff200b12112        172.30.109.59:5100/contrail410-contrail-controller:mainline
"/lib/systemd/syst... 7 weeks ago         Up 2 weeks
controller
6fec888f8145        registry:2                             "/entrypoint.sh /e... 7 weeks ago         Up 2 weeks
registry
root@host-4.1:~# docker exec -it 0ff200b12112 /bin/bash

```

2. Stop Cassandra on all controllers.

```

root@5b5s42:~# service cassandra stop
root@5b5s42:~#
root@5b5s43:~# service cassandra stop
root@5b5s43:~#
root@5b5s44:~# service cassandra stop
root@5b5s44:~#

```

3. Stop Zookeeper on all controllers.

```

root@5b5s42:~# service zookeeper stop
zookeeper stop/waiting
root@5b5s42:~#
root@5b5s43:~# service zookeeper stop
zookeeper stop/waiting
root@5b5s43:~#
root@5b5s44:~# service zookeeper stop
zookeeper stop/waiting
root@5b5s44:~#

```

4. Stop Kafka on all controllers.

```

root@5b5s42:~# service kafka stop
kafka: stopped

```

```
root@5b5s42:~#  
root@5b5s43:~# service kafka stop  
kafka: stopped  
root@5b5s43:~#  
root@5b5s44:~# service kafka stop  
kafka: stopped  
root@5b5s44:~#
```

5. Stop **contrail-hamon** on all controllers, if present.

```
root@5b5s42:~# service contrail-hamon stop  
contrail-hamon stop/waiting  
root@5b5s43:~# service contrail-hamon stop  
contrail-hamon stop/waiting  
root@5b5s44:~# service contrail-hamon stop  
contrail-hamon stop/waiting
```

6. Backup the Zookeeper data directory on all controllers.

```
root@5b5s42:~# cd /var/lib/zookeeper/  
root@5b5s42:/var/lib/zookeeper# cp -R version-2/ version-2-save  
root@5b5s42:/var/lib/zookeeper#  
root@5b5s43:~# cd /var/lib/zookeeper/  
root@5b5s43:/var/lib/zookeeper# cp -R version-2/ version-2-save  
root@5b5s43:/var/lib/zookeeper#  
root@5b5s44:~# cd /var/lib/zookeeper/  
root@5b5s44:/var/lib/zookeeper# cp -R version-2/ version-2-save  
root@5b5s44:/var/lib/zookeeper#
```

7. Backup the Cassandra data directory on all controllers.

```
root@5b5s42:~# cd /var/lib/  
root@5b5s42:/var/lib# cp -R cassandra cassandra-save  
root@5b5s42:/var/lib#  
root@5b5s43:~# cd /var/lib/  
root@5b5s43:/var/lib# cp -R cassandra cassandra-save  
root@5b5s43:/var/lib#  
root@5b5s44:~# cd /var/lib/  
root@5b5s44:/var/lib# cp -R cassandra/ cassandra-save  
root@5b5s44:/var/lib#
```

8. Wipe out the Zookeeper data directory contents on all controllers.

```
root@5b5s42:~# rm -rf /var/lib/zookeeper/version-2/*  
root@5b5s42:~#  
root@5b5s43:~# rm -rf /var/lib/zookeeper/version-2/*  
root@5b5s43:~#  
root@5b5s44:~# rm -rf /var/lib/zookeeper/version-2/*  
root@5b5s44:~#
```

9. Wipe out the Cassandra data directory contents on all controllers.

```
root@5b5s42:~# rm -rf /var/lib/cassandra/*  
root@5b5s42:~#  
root@5b5s43:~# rm -rf /var/lib/cassandra/*  
root@5b5s43:~#  
root@5b5s44:~# rm -rf /var/lib/cassandra/*  
root@5b5s44:~#
```

10. Start Zookeeper on all controllers.

```
root@5b5s42:~# service zookeeper start
zookeeper start/running, process 14180
root@5b5s42:~#
root@5b5s43:~# service zookeeper start
zookeeper start/running, process 11635
root@5b5s43:~#
root@5b5s44:~# service zookeeper start
zookeeper start/running, process 28040
root@5b5s44:~#
```

11. Start Cassandra on all controllers.

```
root@5b5s42:~# service cassandra start
root@5b5s42:~#
root@5b5s43:~# service cassandra start
root@5b5s43:~#
root@5b5s44:~# service cassandra start
root@5b5s44:~#
```

12. Run `python db_json_exim.py --import-from db-dump.json` on any *one* controller.

```
root@5b5s42:~# python db_json_exim.py --import-from db-dump.json
root@5b5s42:~#
```

13. Start **supervisor-config** on all controllers, if present.

```
root@5b5s42:~# service supervisor-config start
supervisor-config start/running, process 19286
root@5b5s42:~#
root@5b5s43:~# service supervisor-config start
supervisor-config start/running, process 28937
root@5b5s43:~#
root@5b5s44:~# service supervisor-config start
supervisor-config start/running, process 21242
root@5b5s44:~#
```

14. Start Kafka on all controllers.

```
root@5b5s42:~# service kafka start
kafka: started
root@5b5s42:~#
root@5b5s43:~# service kafka start
kafka: started
root@5b5s43:~#
root@5b5s44:~# service kafka start
kafka: started
root@5b5s44:~#
```

15. Start **contrail-hamon** on all controllers, if present.

```
root@5b5s42:~# service contrail-hamon start
contrail-hamon start/running, process 1379
root@5b5s42:~#
root@5b5s43:~# service contrail-hamon start
contrail-hamon start/running, process 1230
root@5b5s43:~#
root@5b5s44:~# service contrail-hamon start
```

```
contrail-hamon start/running, process 26843  
root@5b5s44:~#
```

Related •
Documentation

CHAPTER 28

Contrail Application Programming Interfaces (APIs)

- Contrail Analytics Application Programming Interfaces (APIs) and User-Visible Entities (UVEs) on page 709
- Log and Flow Information APIs on page 719
- Working with Neutron on page 726
- Support for Amazon VPC APIs on Contrail OpenStack on page 729

Contrail Analytics Application Programming Interfaces (APIs) and User-Visible Entities (UVEs)

The Contrail **analytics-api** server provides a REST API interface to extract the operational state of the Contrail system.

APIs are used by the Contrail Web user interface to present the operational state to users. Other applications might also use the server's REST APIs for analytics or other uses.

This section describes some of the more common APIs and their uses. To see all of the available APIs, navigate the URL tree at the REST interface, starting at the root **`http://<ip>:<analytics-api-port>`**. You can also view Contrail API information at: <http://configuration-schema-documentation.s3-website-us-west-1.amazonaws.com/R3.2/>

- User-Visible Entities on page 710
- Common UVEs in Contrail on page 711
- Virtual Network UVE on page 711
- Virtual Machine UVE on page 711
- vRouter UVE on page 712
- UVEs for Contrail Nodes on page 712
- Wild Card Query of UVEs on page 713
- Filtering UVE Information on page 713

User-Visible Entities

In Contrail, a User-Visible Entity (UVE) is an object entity that might span multiple components in Contrail and might require aggregation before the complete information of the UVE is presented. Examples of UVEs in Contrail are virtual network, virtual machine, vRouter, and similar objects. Complete operational information for a virtual network might span multiple vRouters, config nodes, control nodes, and the like. The analytics-api server aggregates all of this information through REST APIs.

To get information about a UVE, you must have the UVE type and the UVE key. In Contrail, UVEs are identified by type, such as virtual network, virtual machine, vRouter, and so on. A system-wide unique key is associated with each UVE. The key type could be different, based on the UVE type. For example, perhaps a virtual network uses its name as its UVE key, and in the same system, a virtual machine uses its UUID as its key.

The URL `/analytics/uves` shows the list of all UVE types available in the system.

The following is sample output from `/analytics/uves`:

```
[
{
  href: "http://<system IP>:8081/analytics/uves/xmpp-peers",
  name: "xmpp-peers"
},
{
  href: "http://<system IP>:8081/analytics/uves/service-instances",
  name: "service-instances"
},
{
  href: "http://<system IP>:8081/analytics/uves/config-nodes",
  name: "config-nodes"
},
{
  href: "http://<system IP>:8081/analytics/uves/virtual-machines",
  name: "virtual-machines"
},
{
  href: "http://<system IP>:8081/analytics/uves/bgp-routers",
  name: "bgp-routers"
},
{
  href: "http://<system IP>:8081/analytics/uves/collectors",
  name: "collectors"
},
{
  href: "http://<system IP>:8081/analytics/uves/service-chains",
  name: "service-chains"
},
{
  href: "http://<system IP>:8081/analytics/uves/generators",
  name: "generators"
},
{
  href: "http://<system IP>:8081/analytics/uves/bgp-peers",
  name: "bgp-peers"
},
{
```

```

href: "http://<system IP>:8081/analytics/uves/virtual-networks",
name: "virtual-networks"
},
{
href: "http://<system IP>:8081/analytics/uves/vrouters",
name: "vrouters"
},
{
href: "http://<system IP>:8081/analytics/uves/dns-nodes",
name: "dns-nodes"
}
]

```

Common UVEs in Contrail

This section presents descriptions of some common UVEs in Contrail.

Virtual Network UVE

This UVE provides information associated with a virtual network, such as:

- list of networks connected to this network
- list of virtual machines spawned in this network
- list of access control lists (ACLs) associated with this virtual network
- global input and output statistics
- input and output statistics per virtual network pair

The REST API to get a UVE for a specific virtual network is through HTTP GET, using the URL:

/analytics/uves/virtual-network/<key>

The REST API to get UVEs for all virtual machines is through HTTP GET, using the URL:

/analytics/uves/virtual-networks

Virtual Machine UVE

This UVE provides information associated with a virtual machine, such as:

- list of interfaces in this virtual machine
- list of floating IPs associated with each interface
- input and output statistics

The REST API to get a UVE for a specific virtual machine is through HTTP GET, using the URL:

/analytics/uves/virtual-machine/<key>

The REST API to get UVEs for all virtual machines is through HTTP GET, using the URL:

/analytics/uves/virtual-machines

vRouter UVE

This UVE provides information associated with a vRouter, such as:

- virtual networks present on this vRouter
- virtual machines spawned on the server of this vRouter
- statistics of the traffic flowing through this vRouter

The REST API to get a UVE for a specific vRouter is through HTTP GET, using the URL:

/analytics/uves/vrouter/<key>

The REST API to get UVEs for all virtual machines is through HTTP GET, using the URL:

/analytics/uves/vrouters

UVES for Contrail Nodes

There are multiple node types in Contrail (including the node type vRouter previously described). Other node types include control node, config node, analytics node, and compute node.

There is a UVE for each node type. The common information associated with each node UVE includes:

- the IP address of the node
- a list of processes running on the node
- the CPU and memory utilization of the running processes

Each UVE also has node-specific information, such as:

- the control node UVE has information about its connectivity to the vRouter and other control nodes
- the analytics node UVE has information about the number of generators connected

The REST API to get a UVE for a specific config node is through HTTP GET, using the URL:

/analytics/uves/config-node/<key>

The REST API to get UVEs for all config nodes is through HTTP GET, using the URL:

/analytics/uves/config-nodes



.....
NOTE: Use similar syntax to get UVES for each of the different types of nodes, substituting the node type that you want in place of **config-node**.
.....

Wild Card Query of UVEs

You can use wildcard queries when you want to get multiple UVEs at the same time. Example queries are the following:

The following HTTP GET with wildcard retrieves all virtual network UVEs:

```
/analytics/uves/virtual-network/*
```

The following HTTP GET with wildcard retrieves all virtual network UVEs with name starting with **project1**:

```
/analytics/uves/virtual-network/project1*
```

Filtering UVE Information

It is possible to retrieve filtered UVE information. The following flags enable you to retrieve partial, filtered information about UVEs.

Supported filter flags include:

sfilt : filter by source (usually the hostname of the generator)

mfilt : filter by module (the module name of the generator)

cfilt : filter by content, useful when only part of a UVE needs to be retrieved

kfilt : filter by UVE keys, useful to get multiple, but not all, UVEs of a particular type

Examples

The following HTTP GET with filter retrieves information about virtual network **vn1** as provided by the source **src1**:

```
/analytics/uves/virtual-network/vn1?sfilt=src1
```

The following HTTP GET with filter retrieves information about virtual network **vn1** as provided by all **ApiServer** modules:

```
/analytics/uves/virtual-network/vn1?mfilt=ApiServer
```

Example Output: Virtual Network UVE

Example output for a virtual network UVE:

```
[user@host ~]# curl <system
IP>:8081/analytics/virtual-network/default-domain:demo:front-end | python
-mjson.tool
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  2576  100  2576    0     0  152k      0  --:--:-- --:--:-- --:--:--  157k
{
  "UveVirtualNetworkAgent": {
    "acl": [
      [
        {
          "@type": "string"
        },
      ],
    ],
  },
}
```

```

        "a3s18:VRouterAgent"
    ],
    "in_bytes": {
        "#text": "2232972057",
        "@aggtype": "counter",
        "@type": "i64"
    },
    "in_stats": {
        "@aggtype": "append",
        "@type": "list",
        "list": {
            "@size": "3",
            "@type": "struct",
            "UveInterVnStats": [
                {
                    "bytes": {
                        "#text": "2114516371",
                        "@type": "i64"
                    },
                    "other_vn": {
                        "#text": "default-domain:demo:back-end",
                        "@aggtype": "listkey",
                        "@type": "string"
                    },
                    "tpkts": {
                        "#text": "5122001",
                        "@type": "i64"
                    }
                },
                {
                    "bytes": {
                        "#text": "1152123",
                        "@type": "i64"
                    },
                    "other_vn": {
                        "#text": "__FABRIC__",
                        "@aggtype": "listkey",
                        "@type": "string"
                    },
                    "tpkts": {
                        "#text": "11323",
                        "@type": "i64"
                    }
                }
            ],
            {
                "bytes": {
                    "#text": "8192",
                    "@type": "i64"
                },
                "other_vn": {
                    "#text": "default-domain:demo:front-end",
                    "@aggtype": "listkey",
                    "@type": "string"
                },
                "tpkts": {
                    "#text": "50",
                    "@type": "i64"
                }
            }
        ]
    }
}
]

```

```

    }
  },
  "in_tpkts": {
    "#text": "5156342",
    "@aggtype": "counter",
    "@type": "i64"
  },
  "interface_list": {
    "@aggtype": "union",
    "@type": "list",
    "list": {
      "@size": "1",
      "@type": "string",
      "element": [
        "tap2158f77c-ec"
      ]
    }
  },
  "out_bytes": {
    "#text": "2187615961",
    "@aggtype": "counter",
    "@type": "i64"
  },
  "out_stats": {
    "@aggtype": "append",
    "@type": "list",
    "list": {
      "@size": "4",
      "@type": "struct",
      "UveInterVnStats": [
        {
          "bytes": {
            "#text": "2159083215",
            "@type": "i64"
          },
          "other_vn": {
            "#text": "default-domain:demo:back-end",
            "@aggtype": "listkey",
            "@type": "string"
          },
          "tpkts": {
            "#text": "5143693",
            "@type": "i64"
          }
        }
      ],
      {
        "bytes": {
          "#text": "1603041",
          "@type": "i64"
        },
        "other_vn": {
          "#text": "__FABRIC__",
          "@aggtype": "listkey",
          "@type": "string"
        },
        "tpkts": {
          "#text": "9595",
          "@type": "i64"
        }
      }
    ]
  },
  {

```

```

        "bytes": {
            "#text": "24608",
            "@type": "i64"
        },
        "other_vn": {
            "#text": "__UNKNOWN__",
            "@aggtype": "listkey",
            "@type": "string"
        },
        "tpkts": {
            "#text": "408",
            "@type": "i64"
        }
    },
    {
        "bytes": {
            "#text": "8192",
            "@type": "i64"
        },
        "other_vn": {
            "#text": "default-domain:demo:front-end",
            "@aggtype": "listkey",
            "@type": "string"
        },
        "tpkts": {
            "#text": "50",
            "@type": "i64"
        }
    }
]
}
},
"out_tpkts": {
    "#text": "5134830",
    "@aggtype": "counter",
    "@type": "i64"
},
"virtualmachine_list": {
    "@aggtype": "union",
    "@type": "list",
    "list": {
        "@size": "1",
        "@type": "string",
        "element": [
            "dd09f8c3-32a8-456f-b8cc-fab15189f50f"
        ]
    }
}
},
"UveVirtualNetworkConfig": {
    "connected_networks": {
        "@aggtype": "union",
        "@type": "list",
        "list": {
            "@size": "1",
            "@type": "string",
            "element": [
                "default-domain:demo:back-end"
            ]
        }
    }
},
"routing_instance_list": {

```

```

        "@aggtype": "union",
        "@type": "list",
        "list": {
            "@size": "1",
            "@type": "string",
            "element": [
                "front-end"
            ]
        }
    },
    "total_acl_rules": [
        [
            {
                "#text": "3",
                "@type": "i32"
            },
            ":",
            "a3s14:Schema"
        ]
    ]
}
}

```

Example Output: Example output for a virtual machine UVE:
Virtual Machine UVE

```

[user@host ~]# curl <system
IP>:8081/analytics/virtual-machine/f38eb47e-63d2-4b39-80de-8fe68e6af1e4 |
python -mjson.tool
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                               Dload  Upload  Total  Spent    Left  Speed
100   736   100   736    0    0   160k      0 --:--:-- --:--:-- --:--:--  179k
{
  "UveVirtualMachineAgent": {
    "interface_list": [
      [
        {
          "@type": "list",
          "list": {
            "@size": "1",
            "@type": "struct",
            "VmInterfaceAgent": [
              {
                "in_bytes": {
                  "#text": "2188895907",
                  "@aggtype": "counter",
                  "@type": "i64"
                },
                "in_pkts": {
                  "#text": "5130901",
                  "@aggtype": "counter",
                  "@type": "i64"
                },
                "ip_address": {
                  "#text": "192.168.2.253",
                  "@type": "string"
                },
                "name": {
                  "#text":
"f38eb47e-63d2-4b39-80de-8fe68e6af1e4:ccb085a0-c994-4034-be0f-6fd5ad08ce83",
                  "@type": "string"
                }
              }
            ]
          }
        }
      ]
    ]
  }
}

```

```

    },
    "out_bytes": {
      "#text": "2201821626",
      "@aggtype": "counter",
      "@type": "i64"
    },
    "out_pkts": {
      "#text": "5153526",
      "@aggtype": "counter",
      "@type": "i64"
    },
    "virtual_network": {
      "#text": "default-domain:demo:back-end",
      "@aggtype": "listkey",
      "@type": "string"
    }
  }
}
]
}
},
"a3s19:VRouterAgent"
]
}
}
}

```

Example Output: vRouter UVE

Example output for a vRouter UVE:

```

[user@host ~]# curl <system IP>:8081/analytics/vrouter/a3s18 | python
-mjson.tool
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100    706    100    706    0     0   142k      0  --:--:-- --:--:-- --:--:--   172k
{
  "VrouterAgent": {
    "collector": [
      {
        "#text": "10.xx.17.1",
        "@type": "string"
      },
      "a3s18:VRouterAgent"
    ],
    "connected_networks": [
      {
        "@type": "list",
        "list": {
          "@size": "1",
          "@type": "string",
          "element": [
            "default-domain:demo:front-end"
          ]
        }
      },
      "a3s18:VRouterAgent"
    ],
    "interface_list": [

```

```

[
  {
    "@type": "list",
    "list": {
      "@size": "1",
      "@type": "string",
      "element": [
        "tap2158f77c-ec"
      ]
    }
  },
  "a3s18:VRouterAgent"
],
"virtual_machine_list": [
  [
    {
      "@type": "list",
      "list": {
        "@size": "1",
        "@type": "string",
        "element": [
          "dd09f8c3-32a8-456f-b8cc-fab15189f50f"
        ]
      }
    },
    "a3s18:VRouterAgent"
  ]
],
"xmpp_peer_list": [
  [
    {
      "@type": "list",
      "list": {
        "@size": "2",
        "@type": "string",
        "element": [
          "10.xx.17.2",
          "10.xx.17.3"
        ]
      }
    },
    "a3s18:VRouterAgent"
  ]
]
}

```

- Related Documentation**
- [Juniper Contrail Configuration API Server Documentation](#)
 - [Log and Flow Information APIs on page 719](#)

Log and Flow Information APIs

In Contrail, log and flow analytics information is collected and stored using a horizontally scalable Contrail collector and NoSQL database. The **analytics-api** server provides REST

APIs to extract this information using queries. The queries use well-known SQL syntax, hiding the underlying complexity of the NoSQL tables.

- [HTTP GET APIs on page 720](#)
- [HTTP POST API on page 720](#)
- [POST Data Format Example on page 721](#)
- [Query Types on page 722](#)
- [Examining Query Status on page 722](#)
- [Examining Query Chunks on page 722](#)
- [Example Queries for Log and Flow Data on page 723](#)

HTTP GET APIs

Use the following GET APIs to identify tables and APIs available for querying.

/analytics/tables -- lists the SQL-type tables available for querying, including the hrefs for each of the tables

/analytics/table/<table> -- lists the APIs available to get information for a given table

/analytics/table/<table>/schema -- lists the schema for a given table

HTTP POST API

Use the following POST API information to extract data from a table.

/analytics/query -- format your query using the following SQL syntax:

```
SELECT field1, field2 ...  
FROM table1  
WHERE field1 = value1 AND field2 = value2 ...  
FILTER BY ...  
SORT BY ...  
LIMIT n
```

Additionally, it is mandatory to include the start time and the end time for the data range to define the time period for the query data. The parameters of the query are passed through POST data, using the following fields:

start_time — the start of the time period

end_time — the end of the time period

table — the table from which to extract data

select_fields — the columns to display in the extracted data

where — the list of match conditions

POST Data Format Example

The POST data is in **JSON** format, stored in an **idl** file. A sample file is displayed in the following.



NOTE: The result of the query API is also in JSON format.

```

/*
 * Copyright (c) 2013 Juniper Networks, Inc. All rights reserved.
 */

/*
 * query_rest.idl
 *
 * IDL definitions for query engine REST API
 *
 * PLEASE NOTE: After updating this file, do update json_parse.h
 */

enum match_op {
    EQUAL = 1,
    NOT_EQUAL = 2,
    IN_RANGE = 3,
    NOT_IN_RANGE = 4, // not supported currently
    // following are only for numerical column fields
    LEQ = 5, // column value is less than or equal to filter value
    GEQ = 6, // column value is greater than or equal to filter value
    PREFIX = 7, // column value has the "value" field as prefix
    REGEX_MATCH = 8 // for filters only
}

enum sort_op {
    ASCENDING = 1,
    DESCENDING = 2,
}

struct match {
    1: string name;
    2: string value;
    3: match_op op;
    4: optional string value2; // this is for only RANGE match
}

typedef list<match> term; (AND of match)

enum flow_dir_t {
    EGRESS = 0,
    INGRESS = 1
}

struct query {
    1: string table; // Table to query (FlowSeriesTable, MessageTable,
ObjectVNTTable, ObjectVMTable, FlowRecordTable)
    2: i64 start_time; // Microseconds in UTC since Epoch
    3: i64 end_time; // Microseconds in UTC since Epoch
    4: list<string> select_fields; // List of SELECT fields
    5: list<term> where; // WHERE (OR of terms)
}

```

```
6: optional sort_op sort;
7: optional list<string> sort_fields;
8: optional i32 limit;
9: optional flow_dir_t dir; // direction of flows being queried
10: optional list<match> filter; // filter the processed result by value
}

struct flow_series_result_entry {
  1: optional i64 T; // Timestamp of the flow record
  2: optional string sourcevn;
  3: optional string sourceip;
  4: optional string destvn;
  5: optional string destip;
  6: optional i32 protocol;
  7: optional i32 sport;
  8: optional i32 dport;
  9: optional flow_dir_t direction_ing;
  10: optional i64 packets; // mutually exclusive to 12,13
  11: optional i64 bytes; // mutually exclusive to 12,13
  12: optional i64 sum_packets; // represented as "sum(packets)" in JSON
  13: optional i64 sum_bytes; // represented as "sum(bytes)" in JSON
};

typedef list<flow_series_result_entry> flow_series_result;
```

Query Types

The **analytics-api** supports two types of queries. Both types use the same POST parameters as described in POST API.

- **sync** — Default query mode. The results are sent inline with the query processing.
- **async** — To execute a query in async mode, attach the following header to the POST request: **Expect: 202-accepted**.

Examining Query Status

For an asynchronous query, the **analytics-api** responds with the code: **202 Accepted**. The response contents are a status entity href URL of the form: **/analytics/query/<QueryID>**. The QueryID is assigned by the **analytics-api**. To view the response contents, poll the status entity by performing a GET action on the URL. The status entity has a variable named **progress**, with a number between 0 and 100, representing the approximate percentage completion of the query. When progress is 100, the query processing is complete.

Examining Query Chunks

The status entity has an element named **chunks** that lists portions (chunks) of query results. Each element of this list has three fields: **start_time**, **end_time**, **href**. The **analytics-api** determines how many chunks to list to represent the query data. A chunk can include an empty string ("") to indicate that the data query is not yet available. If a partial result is available, the chunk href is of the form: **/analytics/query/<QueryID>/chunk-partial/<chunk number>**. When the final result of a chunk is available, the href is of the form: **/analytics/query/<QueryID>/chunk-final/<chunk number>**.

Example Queries for Log and Flow Data

The following example query lists the tables available for query.

```
[root@host ~]# curl 127.0.0.1:8081/analytics/tables | python -mjson.tool
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total     Spent    Left     Speed
100    846    100    846      0      0    509k      0 --:--:-- --:--:-- --:--:--   826k
[
  {
    "href": "http://127.0.0.1:8081/analytics/table/MessageTable",
    "name": "MessageTable"
  },
  {
    "href": "http://127.0.0.1:8081/analytics/table/ObjectVNTTable",
    "name": "ObjectVNTTable"
  },
  {
    "href": "http://127.0.0.1:8081/analytics/table/ObjectVMTable",
    "name": "ObjectVMTable"
  },
  {
    "href": "http://127.0.0.1:8081/analytics/table/ObjectVRouter",
    "name": "ObjectVRouter"
  },
  {
    "href": "http://127.0.0.1:8081/analytics/table/ObjectBgpPeer",
    "name": "ObjectBgpPeer"
  },
  {
    "href": "http://127.0.0.1:8081/analytics/table/ObjectRoutingInstance",
    "name": "ObjectRoutingInstance"
  },
  {
    "href": "http://127.0.0.1:8081/analytics/table/ObjectXmppConnection",
    "name": "ObjectXmppConnection"
  },
  {
    "href": "http://127.0.0.1:8081/analytics/table/FlowRecordTable",
    "name": "FlowRecordTable"
  },
  {
    "href": "http://127.0.0.1:8081/analytics/table/FlowSeriesTable",
    "name": "FlowSeriesTable"
  }
]
```

The following example query lists details for the table named **MessageTable**.

```
[root@host ~]# curl 127.0.0.1:8081/analytics/table/MessageTable | python
-mjson.tool
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total     Spent    Left     Speed
100    192    100    192      0      0   102k      0 --:--:-- --:--:-- --:--:--   187k
[
  {
    "href": "http://127.0.0.1:8081/analytics/table/MessageTable/schema",
    "name": "schema"
  },
]
```

```
{
  "href": "http://127.0.0.1:8081/analytics/table/MessageTable/column-values",
  "name": "column-values"
}
]
```

The following example query lists the schema for the table named MessageTable.

```
[root@host ~]# curl 127.0.0.1:8081/analytics/table/MessageTable/schema | python
-mjson.tool
```

```
% Total    % Received % Xferd  Average Speed   Time    Time       Time  Current
                               Dload  Upload    Total   Spent    Left   Speed
100    630    100    630      0      0   275k      0 --:--:-- --:--:-- --:--:--   307k
{
```

```
  "columns": [
    {
      "datatype": "int",
      "index": "False",
      "name": "MessageTS"
    },
    {
      "datatype": "string",
      "index": "True",
      "name": "Source"
    },
    {
      "datatype": "string",
      "index": "True",
      "name": "ModuleId"
    },
    {
      "datatype": "string",
      "index": "True",
      "name": "Category"
    },
    {
      "datatype": "int",
      "index": "True",
      "name": "Level"
    },
    {
      "datatype": "int",
      "index": "False",
      "name": "Type"
    },
    {
      "datatype": "string",
      "index": "True",
      "name": "Messagetype"
    },
    {
      "datatype": "int",
      "index": "False",
      "name": "SequenceNum"
    },
    {
      "datatype": "string",
      "index": "False",
      "name": "Context"
    }
  ]
}
```

```

    },
    {
      "datatype": "string",
      "index": "False",
      "name": "Xmlmessage"
    }
  ],
  "type": "LOG"
}

```

The following set of example queries explore a message table.

```

root@a6s45:~# cat filename
{ "end_time": "now" , "select_fields": ["MessageTS", "Source", "ModuleId",
"Category", "Messagetype", "SequenceNum", "Xmlmessage", "Type", "Level",
"NodeType", "InstanceId"] , "sort": 1 , "sort_fields": ["MessageTS"] ,
"start_time": "now-10m" , "table": "MessageTable" , "where": {"name": "ModuleId",
"value": "contrail-control", "op": 1, "suffix": null, "value2": null}, {"name":
"Messagetype", "value": "BGPRouterInfo", "op": 1, "suffix": null, "value2": null}
}

root@a6s45:~#
root@a6s45:~# curl -X POST --data @filename 127.0.0.1:8081/analytics/query --header
"Content-Type:application/json" | python -mjson.tool
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  9765    0  9297  100    468    9168    461  0:00:01  0:00:01 --:--:--  9177
{
  "value": [
    {
      "Category": null,
      "InstanceId": "0",
      "Level": 2147483647,
      "MessageTS": 1428442589947392,
      "Messagetype": "BGPRouterInfo",
      "ModuleId": "contrail-control",
      "NodeType": "Control",
      "SequenceNum": 1302,
      "Source": "a6s45",
      "Type": 6,
      "Xmlmessage": "<BGPRouterInfo type=''><data
type=''><BgpRouterState><name type=''
>a6s45</name><cpu_info type=''><CpuLoadInfo><num_cpu type=''>4</num_cpu
><meminfo type=''><MemInfo><virt type=''>438436</virt><peakvirt type=''
>561048</peakvirt><res type=''>12016</res></MemInfo></meminfo><cpu_share
type=''>0.0416667</cpu_share></CpuLoadInfo></cpu_info><cpu_share type=''
>0.0416667</cpu_share></BgpRouterState></data></BGPRouterInfo>"
    },
    {
      "Category": null,
      "InstanceId": "0",
      "Level": 2147483647,

```

Working with Neutron

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

- [Data Structure on page 726](#)
- [Network Sharing in Neutron on page 726](#)
- [Commands for Neutron Network Sharing on page 727](#)
- [Support for Neutron APIs on page 727](#)
- [Contrail Neutron Plugin on page 728](#)
- [DHCP Options on page 728](#)
- [Incompatibilities on page 729](#)

Data Structure

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

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

- reads a list of resources (for example. **GET /virtual-networks**), then
- iterates and reads in the details of the resource (**GET /virtual-network/<uuid>**).

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

The following features in Contrail improve performance with Neutron:

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

Network Sharing in Neutron

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

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

Using the Shared Attribute

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

neutron net-list --shared

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

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

Using the Router:External Attribute

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

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

then associating the IP address pool with their instances.



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

Commands for Neutron Network Sharing

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

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

Support for Neutron APIs

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

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

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

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

Contrail Neutron Plugin

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

- Network
- Subnet
- Port

It also implements the following standard and upstreamed Neutron extensions:

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

The following Contrail-specific extensions are implemented:

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

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

DHCP Options

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

Example

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

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

Incompatibilities

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

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

Support for Amazon VPC APIs on Contrail OpenStack

- [Overview of Amazon Virtual Private Cloud on page 729](#)
- [Mapping Amazon VPC Features to OpenStack Contrail Features on page 730](#)
- [VPC and Subnets Example on page 730](#)
- [Euca2ools CLI for VPC and Subnets on page 731](#)
- [Security in VPC: Network ACLs Example on page 732](#)
- [Euca2ools CLI for Network ACLs on page 733](#)
- [Security in VPC: Security Groups Example on page 733](#)
- [Euca2ools CLI for Security Groups on page 734](#)
- [Elastic IPs in VPC on page 734](#)
- [Euca2ools CLI for Elastic IPs on page 735](#)
- [Euca2ools CLI for Route Tables on page 735](#)
- [Supported Next Hops on page 735](#)
- [Internet Gateway Next Hop Euca2ools CLI on page 736](#)
- [NAT Instance Next Hop Euca2ools CLI on page 736](#)
- [Example: Creating a NAT Instance with Euca2ools CLI on page 736](#)

Overview of Amazon Virtual Private Cloud

The current Grizzly release of OpenStack supports Elastic Compute Cloud (EC2) API translation to OpenStack Nova, Quantum, and Keystone calls. EC2 APIs are used in Amazon Web Services (AWS) and virtual private clouds (VPCs) to launch virtual machines, assign IP addresses to virtual machines, and so on. A VPC provides a container where applications can be launched and resources can be accessed over the networking services provided by the VPC.

Contrail enhances its use of EC2 APIs to support the Amazon VPC APIs.

The Amazon VPC supports networking constructs such as: subnets, DHCP options, elastic IP addresses, network ACLs, security groups, and route tables. The Amazon VPC APIs

are now supported on the Openstack Contrail distribution, so users of the Amazon EC2 APIs for their VPC can use the same scripts to move to an Openstack Contrail solution.

Euca2ools are command-line tools for interacting with Amazon Web Services (AWS) and other AWS-compatible web services, such as OpenStack. **Euca2ools** have been extended in OpenStack Contrail to add support for the Amazon VPC, similar to the support that already exists for the Amazon EC2 CLI.

For more information about Amazon VPC and AWS EC2, see:

- Amazon VPC documentation:
http://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/VPC_Introduction.html
- Amazon VPC API list:
<http://docs.aws.amazon.com/AWSEC2/latest/APIReference/query-apis.html>

Mapping Amazon VPC Features to OpenStack Contrail Features

The following table compares Amazon VPC features to their equivalent features in OpenStack Contrail.

Table 90: Amazon VPC and OpenStack Contrail Feature Comparison

Amazon VPC Feature	OpenStack Contrail Feature
VPC	Project
Subnets	Networks (Virtual Networks)
DHCP options	IPAM
Elastic IP	Floating IP
Network ACLs	Network ACLs
Security Groups	Security Groups
Route Table	Route Table

VPC and Subnets Example

When creating a new VPC, the user must provide a classless inter-domain routing (CIDR) block of which all subnets in this VPC will be part.

In the following example, a VPC is created with a CIDR block of 10.1.0.0/16. A subnet is created within the VPC CIDR block, with a CIDR block of 10.1.1.0/24. The VPC has a default network ACL named **acl-default**.

All subnets created in the VPC are automatically associated to the default network ACL. This association can be changed when a new network ACL is created. The last command

in the list below creates a virtual machine using the image **ami-00000003** and launches with an interface in **subnet-5eb34ed2**.

```
# euca-create-vpc 10.1.0.0/16
VPC VPC:vpc-8352aa59 created

# euca-describe-vpcs
VpcId      CidrBlock      DhcpOptions
-----
vpc-8352aa59  10.1.0.0/16    None

# euca-create-subnet -c 10.1.1.0/24 vpc-8352aa59
Subnet: subnet-5eb34ed2 created

# euca-describe-subnets
Subnet-id   Vpc-id         CidrBlock
-----
subnet-5eb34ed2 vpc-8352aa59  10.1.1.0/24

# euca-describe-network-acls
AclId
-----
acl-default(def)
vpc-8352aa59

      Rule   Dir   Action  Proto  Port  Range  Cidr
      ----   --   -
      100    ingress allow   -1     0    65535  0.0.0.0/0
      100    egress  allow   -1     0    65535  0.0.0.0/0
      32767  ingress deny    -1     0    65535  0.0.0.0/0
      32767  egress  deny    -1     0    65535  0.0.0.0/0

      Association      SubnetId      AclId
      -----
      aclassoc-0c549d66  subnet-5eb34ed2  acl-default

# euca-run-instances -s subnet-5eb34ed2 ami-00000003
```

Euca2ools CLI for VPC and Subnets

The following **euca2ools** CLI commands are used to create, define, and delete VPCs and subnets:

- **euca-create-vpc**
- **euca-delete-vpc**
- **euca-describe-vpcs**
- **euca-create-subnet**
- **euca-delete-subnet**
- **euca-describe-subnets**

Security in VPC: Network ACLs Example

Network ACLs support ingress and egress rules for traffic classification and filtering. The network ACLs are applied at a subnet level.

In the following example, a new ACL, **acl-ba7158**, is created and an existing subnet is associated to the new ACL.

```
# euca-create-network-acl vpc-8352aa59
acl-ba7158c
```

```
# euca-describe-network-acls
```

```
AclId
```

```
-----
```

```
acl-default(def)
```

```
vpc-8352aa59
```

Rule	Dir	Action	Proto	Port	Range	Cidr
100	ingress	allow	-1	0	65535	0.0.0.0/0
100	egress	allow	-1	0	65535	0.0.0.0/0
32767	ingress	deny	-1	0	65535	0.0.0.0/0
32767	egress	deny	-1	0	65535	0.0.0.0/0

Association	SubnetId	AclId
aclassoc-0c549d66	subnet-5eb34ed2	acl-default

```
AclId
```

```
-----
```

```
acl-ba7158c
```

```
vpc-8352aa59
```

Rule	Dir	Action	Proto	Port	Range	Cidr
32767	ingress	deny	-1	0	65535	0.0.0.0/0
32767	egress	deny	-1	0	65535	0.0.0.0/0

```
# euca-replace-network-acl-association -a aclassoc-0c549d66 acl-ba7158c
aclassoc-0c549d66
```

```
# euca-describe-network-acls
```

```
AclId
```

```
-----
```

```
acl-default(def)
```

```
vpc-8352aa59
```

Rule	Dir	Action	Proto	Port	Range	Cidr
100	ingress	allow	-1	0	65535	0.0.0.0/0
100	egress	allow	-1	0	65535	0.0.0.0/0
32767	ingress	deny	-1	0	65535	0.0.0.0/0
32767	egress	deny	-1	0	65535	0.0.0.0/0

Association	SubnetId	AclId
-----	-----	-----

```
AclId
```

```

-----
acl-ba7158c
vpc-8352aa59

```

Rule	Dir	Action	Proto	Port	Range	Cidr
32767	ingress	deny	-1	0	65535	0.0.0.0/0
32767	egress	deny	-1	0	65535	0.0.0.0/0

Association	SubnetId	AclId
aclassoc-0c549d66	subnet-5eb34ed2	acl-ba7158c

Euca2ools CLI for Network ACLs

The following **euca2ools** CLI commands are used to create, define, and delete VPCs and subnets:

- **euca-create-network-acl**
- **euca-delete-network-acl**
- **euca-replace-network-acl-association**
- **euca-describe-network-acls**
- **euca-create-network-acl-entry**
- **euca-delete-network-acl-entry**
- **euca-replace-network-acl-entry**

Security in VPC: Security Groups Example

Security groups provide virtual machine level ingress/egress controls. Security groups are applied to virtual machine interfaces.

In the following example, a new security group is created. The rules can be added or removed for the security group based on the commands listed for **euca2ools**. The last line launches a virtual machine using the newly created security group.

```
# euca-describe-security-groups
```

GroupId	VpcId	Name	Description
sg-6d89d7e2	vpc-8352aa59	default	

Direction	Proto	Start	End	Remote
Ingress	any	0	65535	[0.0.0.0/0]
Egress	any	0	65535	[0.0.0.0/0]

```
# euca-create-security-group -d "TestGroup" -v vpc-8352aa59 testgroup
GROUP sg-c5b9d22a testgroup TestGroup
```

```
# euca-describe-security-groups
```

```

GroupId      VpcId      Name      Description
-----
sg-6d89d7e2  vpc-8352aa59  default

```

Direction	Proto	Start	End	Remote
Ingress	any	0	65535	[0.0.0.0/0]
Egress	any	0	65535	[0.0.0.0/0]

```

GroupId      VpcId      Name      Description
-----
sg-c5b9d22a  vpc-8352aa59  testgroup  TestGroup

```

Direction	Proto	Start	End	Remote
Egress	any	0	65535	[0.0.0.0/0]

```

# euca-run-instances -s subnet-5eb34ed2 -g testgroup ami-00000003

```

Euca2ools CLI for Security Groups

The following **euca2ools** CLI commands are used to create, define, and delete security groups:

- **euca-create-security-group**
- **euca-delete-security-group**
- **euca-describe-security-groups**
- **euca-authorize-security-group-egress**
- **euca-authorize-security-group-ingress**
- **euca-revoke-security-group-egress**
- **euca-revoke-security-group-ingress**

Elastic IPs in VPC

Elastic IPs in VPCs are equivalent to the floating IPs in the Contrail Openstack solution.

In the following example, a floating IP is requested from the system and assigned to a particular virtual machine. The prerequisite is that the provider or Contrail administrator has provisioned a network named “public” and allocated a floating IP pool to it. This “public” floating IP pool is then internally used by the tenants to request public IP addresses that they can use and attach to virtual machines.

```

# euca-allocate-address --domain vpc
ADDRESS 10.84.14.253    eipalloc-78d9a8c9

# euca-describe-addresses --filter domain=vpc
Address      Domain      AllocationId      InstanceId(AssociationId)
-----
10.84.14.253  vpc        eipalloc-78d9a8c9

```

```
# euca-associate-address -a eipalloc-78d9a8c9 i-00000008
ADDRESS eipassoc-78d9a8c9

# euca-describe-addresses --filter domain=vpc
Address      Domain    AllocationId      InstanceId(AssociationId)
-----
10.84.14.253  vpc       eipalloc-78d9a8c9  i-00000008(eipassoc-78d9a8c9)
```

Euca2ools CLI for Elastic IPs

The following **euca2ools** CLI commands are used to create, define, and delete elastic IPs:

- **euca-allocate-address**
- **euca-release-address**
- **euca-describe-addresses**
- **euca-associate-address**
- **euca-disassociate-address**

Euca2ools CLI for Route Tables

Route tables can be created in an Amazon VPC and associated with subnets. Traffic exiting a subnet is then looked up in the route table and, based on the route lookup result, the next hop is chosen.

The following **euca2ools** CLI commands are used to create, define, and delete route tables:

- **euca-create-route-table**
- **euca-delete-route-table**
- **euca-describe-route-tables**
- **euca-associate-route-table**
- **euca-disassociate-route-table**
- **euca-replace-route-table-association**
- **euca-create-route**
- **euca-delete-route**
- **euca-replace-route**

Supported Next Hops

The supported next hops for the current release are:

- Local Next Hop

Designating local next hop indicates that all subnets in the VPC are reachable for the destination prefix.

- Internet Gateway Next Hop

This next hop is used for traffic destined to the Internet. All virtual machines using the Internet gateway next hop are required to use an Elastic IP to reach the Internet, because the subnet IPs are private IPs.

- NAT instance

To create this next hop, the user needs to launch a virtual machine that provides network address translation (NAT) service. The virtual machine has two interfaces: one internal and one external, both of which are automatically created. The only requirement here is that a “public” network should have been provisioned by the admin, because the second interface of the virtual machine is created in the “public” network.

Internet Gateway Next Hop Euc2ools CLI

The following **euc2ools** CLI commands are used to create, define, and delete Internet gateway next hop:

- **euc2-attach-internet-gateway**
- **euc2-create-internet-gateway**
- **euc2-delete-internet-gateway**
- **euc2-describe-internet-gateways**
- **euc2-detach-internet-gateway**

NAT Instance Next Hop Euc2ools CLI

The following **euc2ools** CLI commands are used to create, define, and delete NAT instance next hops:

- **euc2-run-instances**
- **euc2-terminate-instances**

Example: Creating a NAT Instance with Euc2ools CLI

The following example creates a NAT instance and creates a default route pointing to the NAT instance.

```
# euc2-describe-route-tables
RouteTableId    Main    VpcId    AssociationId    SubnetId
-----
rtb-default     yes     vpc-8352aa59    rtbassoc-0c549d66    subnet-5eb34ed2

                Prefix    NextHop
                -----
                10.1.0.0/16    local

# euc2-describe-images
IMAGE    ami-00000003    None (ubuntu)    2c88a895fdea4461a81e9b2c35542130
IMAGE    ami-00000005    None (nat-service)    2c88a895fdea4461a81e9b2c35542130

# euc2-run-instances ami-00000005
```



```
# euca-create-route --cidr 0.0.0.0/0 -i i-00000006 rtb-default
```

```
# euca-describe-route-tables
```

RouteTableId	Main	VpcId	AssociationId	SubnetId
-----	----	-----	-----	-----
rtb-default	yes	vpc-8352aa59	rtbassoc-0c549d66	subnet-5eb34ed2

Prefix	NextHop
-----	-----
10.1.0.0/16	local
0.0.0.0/0	i-00000006

