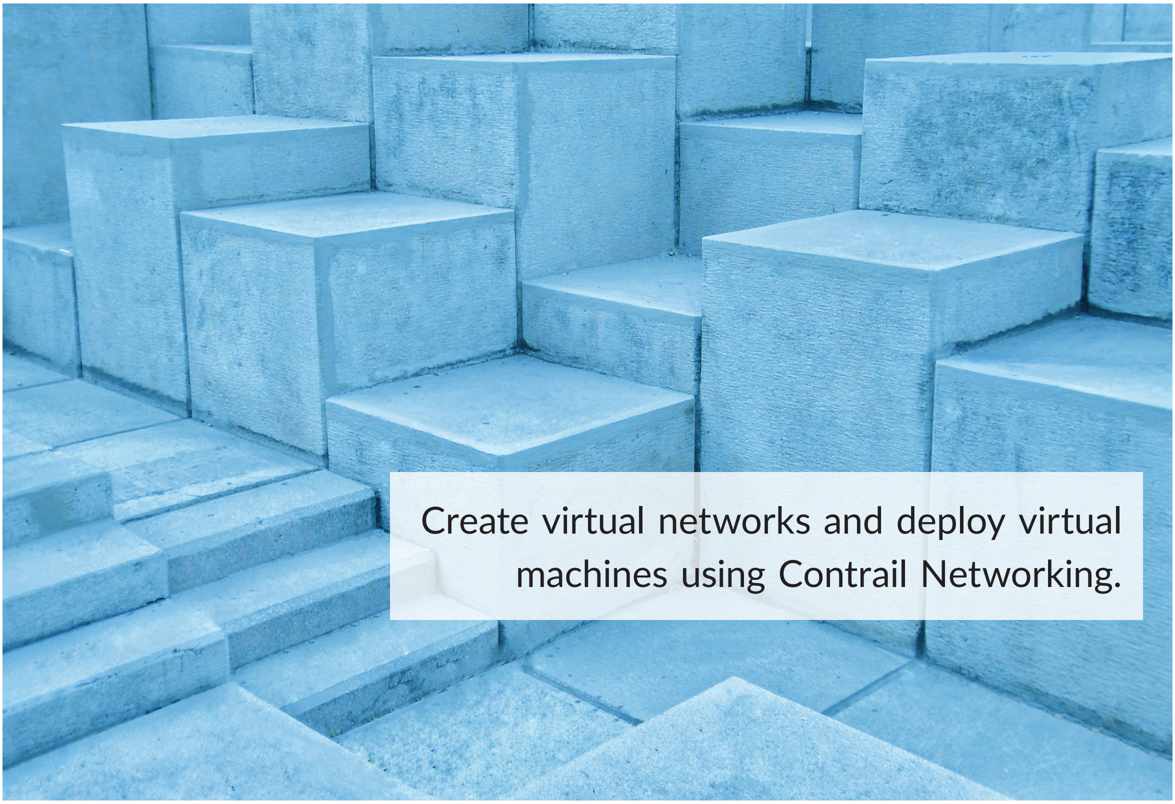
DAY ONE: CONTRAIL NETWORKING UP AND RUNNING WITH OPENSTACK



By Kolachalam Krishna Kishore

DAY ONE: CONTRAIL UP AND RUNNING WITH OPENSTACK

Starting from scratch, this book provides network administrators with a roadmap of how to set up and operate a Contrail SDN-based cloud. It supplies everything you need to know to boot Contrail Networking and then quickly create your first cluster using the most common configurations. Written and reviewed by practicing JTAC engineers, this Day One book includes follow-along configurations, detailed illustrations, and lab advice with useful links to Contrail Networking documentation. So get your cloud up and running on day one.

"This is a perfect Day One book to get you started with Contrail Networking. Following a brief tour of terms, the book helps you install, verify, and then get up and running building your first cloud. The book contains Contrail insights along the way and is authored and reviewed by Juniper JTAC engineers."

Payum Moussavi, VP Technical Support, Juniper Networks

"At last, a thorough up and running book devoted to Contrail Networking. Build your first cloud on day one! Everything you need plus insights from Kishore and his JTAC technical reviewers."

Raghupathi C., DIrector of Technical Support, Juniper Networks

IT'S DAY ONE AND YOU HAVE A JOB TO DO, SO LEARN ABOUT:

- The concept of the cloud and its role in a data center.
- The basics of OpenStack and its services.
- The various functional blocks of Contrail Networking architecture and how those blocks interact with each other.
- How to install Contrail Networking with OpenStack Kolla using an Ansible method.
- The working dashboards of OpenStack and Contrail Networking and how to use them.
- How to create virtual networks and deploy VMs connected to them.
- How to apply security policies restricting access to critical services.
- Service chaining and BGPaaS concepts by deploying a three-tier web application.

Juniper Networks Books are focused on network reliability and efficiency.

Peruse the complete library at www.juniper.net/dayone.

JUNIPER NETWORKS

Day One: Contrail Networking Up and Running with OpenStack

by Kolachalam Krishna Kishore

Juniper Networks

Published by Juniper Networks Books
Authors: Kolachalam Krishna Kishore
Technical Reviewers: Rahul V, Robin K, Ramakantha Guthi, Pramod N, Anish KT, and Aiman I
Editor in Chief: Patrick Ames
Copy Editor: Nancy Koerbel

Printed in the USA by Vervante Books.

Version History: v1, March. 2021
2 3 4 5 6 7 8 9 10
Comments, errata: dayone@juniper.net

About the Author
Kolachalam Krishna Kishore is a technical support engineer at Juniper Networks with 14+ years of experience in the networking industry and JNCIE security certification JNCIE#425. Aside from supporting Juniper customers, he enjoys tinkering and writing scripts in Python or Bash.

Author's Acknowledgments
I would like to thank Patrick Ames for guidance and for providing the opportunity to write this book. Also, I would like to acknowledge and thank my manager, Brahmeswara Reddy K, and my previous manager Arun K S, who encouraged and supported me during this journey. My appreciation to Rahul V, Robin K, Ramakantha Guthi, Pramod N, Anish KT and Aiman I for their detailed technical reviews. And I would also like to thank my wife, Rakshita, for her support.

Welcome to Day One

This book is part of the *Day One* library, produced and published by Juniper Networks Books. *Day One* books cover the Junos OS and Juniper Networks network administration with straightforward explanations, step-by-step instructions, and practical examples that are easy to follow.

- Download a free PDF edition at https://www.juniper.net/dayone
- Purchase the paper edition at Vervante Books (www.vervante.com).

Key Contrail Networking Resources

The Juniper TechLibrary supports Contrail Networking with an entire suite of excellent documentation. This book is not a substitute for that body of work. The author assumes you have reviewed the documentation: https://www.juniper.net/documentation/product/en_US/contrail-networking/19/.

Before Reading This Book You Need

- An understanding of the architecture of Contrail / Tungsten Fabric: https://tungstenfabric.github.io/website/Tungsten-Fabric-Architecture.html.
- A working understanding of the Linux CLI.
- Administrative knowledge of Linux virtualization using KVM/Qemu.
- Knowledge of the basics of Ansible.
- An understanding of the networking and routing protocol BGP.
- Knowledge of how to access to the internet from bare metal servers to download packages and Docker containers.

After Reading This Book You Will...

- Understand the concept of the cloud and its role in a data center.
- Understand the basics of OpenStack and its services.
- Understand various functional blocks of Contrail Networking architecture and how those blocks interact with each other.
- Understand how to install Contrail Networking with OpenStack Kolla using an Ansible method.
- Know the working dashboards of OpenStack and Contrail Networking and how to use them.
- Be able to create virtual networks and deploy VMs connected to them.
- Apply security policies restricting access to critical services.
- Understand service chaining and BGPaaS concepts and deploy a three-tier web application.

Chapter 1

Introduction To Cloud and Its Terminology

This chapter reviews the concept of cloud in relation to data centers; it addresses types of clouds, the role of an orchestrator in cloud, and understanding the role of Juniper Contrail in data center clouds.

So what is cloud? Cloud computing is the availability of on-demand computer resources such as CPU, memory, network, and storage without active management by the user. Virtualization is a key technology that enables cloud computing, a term generally used to describe data centers (DC) available to many users over the internet or a corporate intranet.

Standard DC

When an application or workload is hosted on a server, it takes several teams and man hours to provision it manually. Some of the steps involved are:

- Allotting an IP address
- Determining the rack and switch port where the server needs to be connected
- Installing an OS
- Connecting storage
- Setting up the application
- Creating required firewall rules and load balancer rules

Though this list is not comprehensive, it shows even a simple task like hosting an application can have many steps carried out by different teams.

Cloud-enabled DC

Cloud-enabled DC is a term referring to the process of pooling servers (compute power), networking, and storage to host applications on demand. This pool of resources is presented as a *single pane of glass* or a single view.

Several automation tools—such as HEAT (an orchestration tool of OpenStack), Ansible, Puppet, and Chef—help in automating the provision steps just mentioned.

For example, a developer or a DevOps engineer who wishes to host an application on cloud can make use of tools like HEAT, Ansible, and simple, readable YAML files to let the automation take care of the actual deployment. You can complete the base line provisioning of the server and set up the application in a few minutes rather than waiting for all other teams to complete their share of steps.

What is Coud Made Of?

Compute (servers), networks, and storage hardware are the primary building blocks of cloud. Since cloud operators share the cloud with many tenants, it is imperative to use virtualization techniques in all three areas mentioned above and to share the resources as defined by the cloud operator.

Servers are virtualized using Hypervisors like QEMU/KVM, ESXi, XEN, etc.

Networks are virtualized using encapsulation like VxLAN, MPLSoGRE, MPLSoUDP, etc.

Benefits of Cloud

Some of the advantages of cloud are self-service, cost and efficiency, time to market, scalability, and more. Let's review.

Cost and efficiency: Since most clouds implement virtualization, the cost of running a service on a server is significantly reduced. In a traditional network, you would install one application on one physical server with its supported OS and libraries. However, in cloud environments this required compute power and RAM would be carved out of a physical server, which can be referred as a Virtual Machine (VM) or *instance*. This VM can run its own OS, libraries, and applications independent of the host or other VMs hosted on the Bare Metal Server (BMS).

Time to market: As with most of the steps involved in bringing up a server, provisioning the networks and associated storage, then applying required security, is automated. The time required to get an application up and running to the end user is reduced from a few days, or even a few weeks, to a few hours.

Scalability: Cloud orchestration and SDN controllers can monitor the defined parameters. They can also be programmed to spin up or shut down virtual servers when the load increases or decreases.

Types of Cloud

Public

A public cloud refers to a pool of servers set up by a cloud provider and offered to several customers. Customers can access their services hosted on a public cloud over the internet. Reduced overall cost, fewer management and maintenance costs, and scalability are some of the advantages of public clouds. Vendor lock-in, data security, and compliance are some of the disadvantages of public clouds.

Private

When an enterprise or an entity sets up its own cloud using tools like OpenStack and Contrail Networking on company-owned servers it's called a private cloud.

Hybrid

Hybrid clouds combine the best usages of public and private clouds.

Multi-cloud

A multi-cloud allows you to connect any kind of cloud (public, private, VMware-based) and any kind of workload (BMS, VMs, containers, physical network devices) and then integrate them into any kind of environment or deployment (greenfield or brownfield, including multivendor setups).

Service Models

Infrastructure-as-a-Service

The infrastructure-as-a-service (IaaS) model describes the concept of abstracting and virtualizing underlying infrastructure of compute, network, storage, scaling, and security, through high level APIs. The APIs are provided by orchestration tools such as OpenStack, CloudStack, or VMware vCenter.

IaaS service providers supply these resources from their large pool of devices installed in their data centers.

Compute resources are virtualized using hypervisors where several VMs can be hosted on single bare metal server, or through the use of containers where networks are virtualized using concepts such as VLAN, VxLAN, and VRFs.

The NIST definition of cloud computing describes IaaS as "*where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications. The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, and deployed applications; and possibly limited control of select networking components (e.g., host firewalls).*"

Examples of IaaS clouds (Figure 1.1) are Amazon Web Services, Google Compute, and Microsoft Azure.

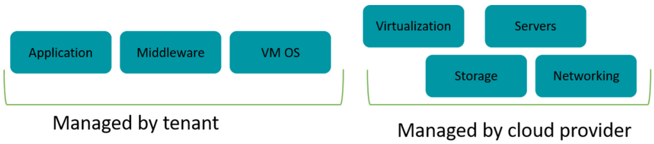


Figure 1.1 Infrastructure as a Service Model

Platform-as-a-Service

Platform-as-a-service (PaaS) provides a standard development environment including programming language execution environment, database, and web server. In this model, the cloud user does not have control over the underlying operating system or system resources on which the environment is hosted.

Examples of PaaS (Figure 1.2) are AWS Elastic Beanstalk, Windows Azure, and Apache Stratos.

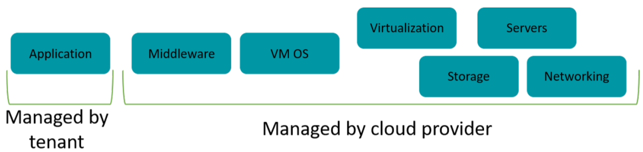


Figure 1.2 Platform as a Service Model

Software as a Service

Software as a service (SaaS) offers a complete software solution provided on a pay-as-you-go basis (Figure 1.3). Examples are Google Apps and SalesForce.

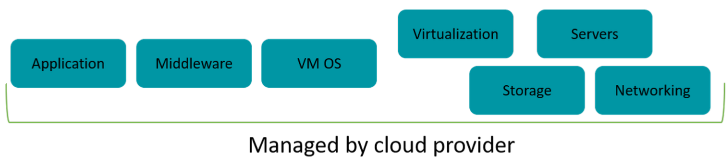


Figure 1.3 Software as a Service Model

What is SDN?

Software defined networking (SDN) separates the network control plane from the forwarding plane to enable more automated provisioning and policy-based management of network resources as shown in Figure 1.4.

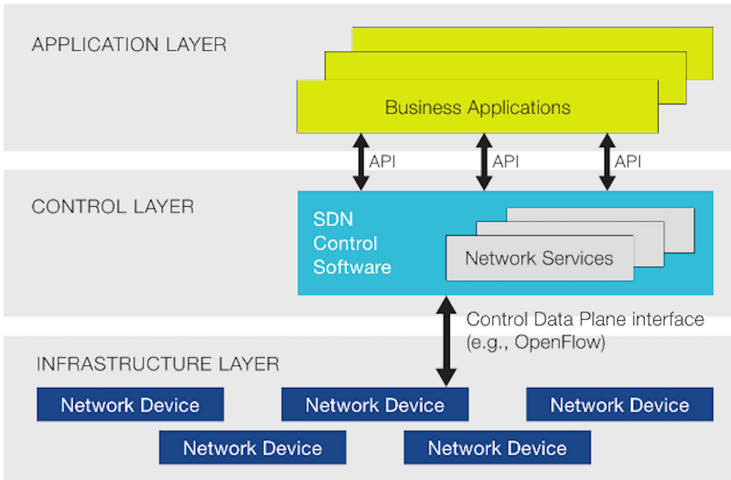


Figure 1.4 *SDN Architecture*

The origins of SDN can be traced to a research collaboration between Stanford University and the University of California at Berkeley that ultimately created the OpenFlow protocol around 2008.

The fundamental idea behind SDN is to program the network rather than configure it by separating the control plane management of network devices from the underlying data plane that forwards network traffic. SDN also centralizes and abstracts control and automates workflows across many places in the network.

IDC defines SDN as *"Datacenter SDN architectures feature software-defined overlays or controllers that are abstracted from the underlying network hardware, offering intent-or policy-based management of the network as a whole. This results in a data center network that is better aligned with the needs of application workloads through automated (thereby faster) provisioning, programmatic network management, pervasive application-oriented visibility, and where needed, direct integration with cloud orchestration platforms."*

Since one of the primary approaches of SDN is to automate provisioning, it plays well in a cloud environment where the orchestrator presents the SDN controller with the intent of what is to be provisioned. The SDN controller compiles this into a set of instructions or configuration lines that can be applied to the forwarding plane to realize the intent.

A Brief Introduction to OpenStack

Per the description on the OpenStack website, *it is an open-source software to control large pool of compute, storage and networking resources throughout a data center*. Some other definitions also refer to it as an Operating System of cloud. These resources can be managed through a dashboard, an OpenStack Client (CLI), or via an Application Programming Interface (API).

OpenStack consists of several inter-related projects, also referred to as *services* that operate and manage different aspects of the cloud environment. For example, Nova manages the compute; Neutron controls and manages the network.

Each OpenStack project exposes its functionality through APIs. These APIs can be used to build a custom dashboard for managing the cloud or you can also use OpenStack dashboard service Horizon for the same purpose as shown in Figure 1.5.

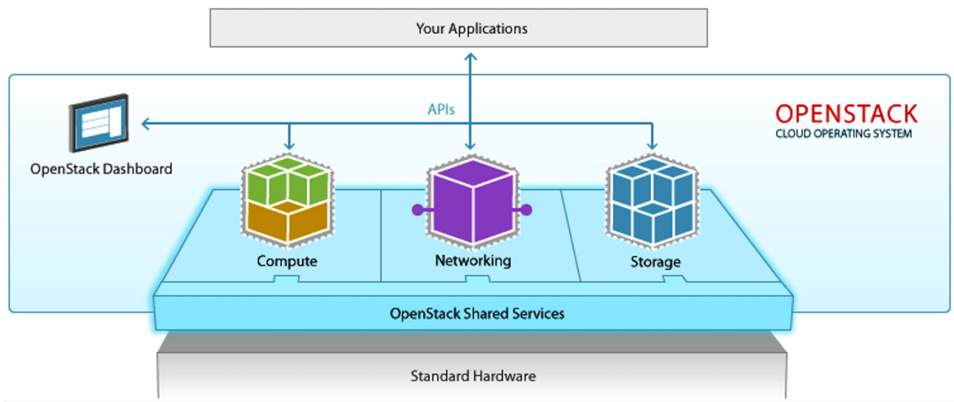
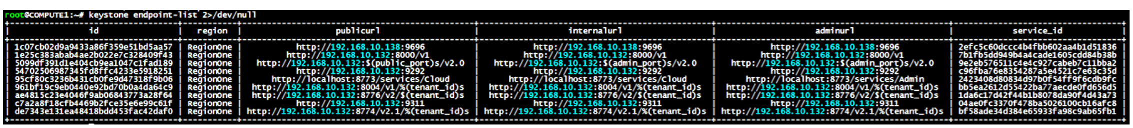


Figure 1.5 OpenStack Overview

OpenStack Services

Keystone

Each time an API request is sent to any service in OpenStack it must contain a token that Keystone uses to authenticate the user. It also provides a list of services that are available in a setup and their location using URLs as shown in Figure 1.6.



root@COMPUTE1:~# keystone endpoint-list 2>/dev/null

id	region	publicurl	internalurl	adminurl	service_id
1c07cb02d9a9433a86f359e51bd5aa57	RegionOne	http://192.168.10.138:9696	http://192.168.10.138:9696	http://192.168.10.138:9696	2efc5c60dccc4b4fbb602aa4b1d51836
1e25c383abab4ae2b022e7c328409f43	RegionOne	http://192.168.10.132:8000/v1	http://192.168.10.132:8000/v1	http://192.168.10.132:8000/v1	7b1fb5dd949b4a4cade1605cdd84b38b
5099df391d1e404cb9ea1047c1fad189	RegionOne	http://192.168.10.132:$(public_port)s/v2.0	http://192.168.10.132:$(admin_port)s/v2.0	http://192.168.10.132:$(admin_port)s/v2.0	9e2eb576511c4e4c927cabeb7c11bba2
54702506987345fd8ffc4233e5918251	RegionOne	http://192.168.10.132:9292	http://192.168.10.132:9292	http://192.168.10.132:9292	c96fba76e8354287a5e4521c7e63c35d
95cf80c3236b431cb0fe9d47318f9b06	RegionOne	http://localhost:8773/services/Cloud	http://localhost:8773/services/Cloud	http://localhost:8773/services/Admin	2423408d80834d97b0f54ff9f6cdb9fc
961bf19c9eb0440e92bd70b0a4da64c9	RegionOne	http://192.168.10.132:8004/v1/%(tenant_id)s	http://192.168.10.132:8004/v1/%(tenant_id)s	http://192.168.10.132:8004/v1/%(tenant_id)s	bb5ea2612d55422ba77aecde0fd656d5
ae4815c23e4046f9ab06843773a28f64	RegionOne	http://192.168.10.132:8776/v2/$(tenant_id)s	http://192.168.10.132:8776/v2/$(tenant_id)s	http://192.168.10.132:8776/v2/$(tenant_id)s	1da6c17d42f44b1b8078da90f4d43a73
c7a2a8f18cfb4469b2fce35e6e99c61f	RegionOne	http://192.168.10.132:9311	http://192.168.10.132:9311	http://192.168.10.132:9311	04ae0fc3370f478ba5026100cb16afc8
de7343e131ea48418bdd453fac42daf0	RegionOne	http://192.168.10.132:8774/v2.1/%(tenant_id)s	http://192.168.10.132:8774/v2.1/%(tenant_id)s	http://192.168.10.132:8774/v2.1/%(tenant_id)s	bf58ade34d384e65933fa98c9ab65fb1

Figure 1.6 Keystone Endpoint List

Horizon

The default UI interface that comes with OpenStack is Horizon. It can be used to create, delete, or update networks, instances, users, project (tenants), or to upload images as shown in Figure 1.7.

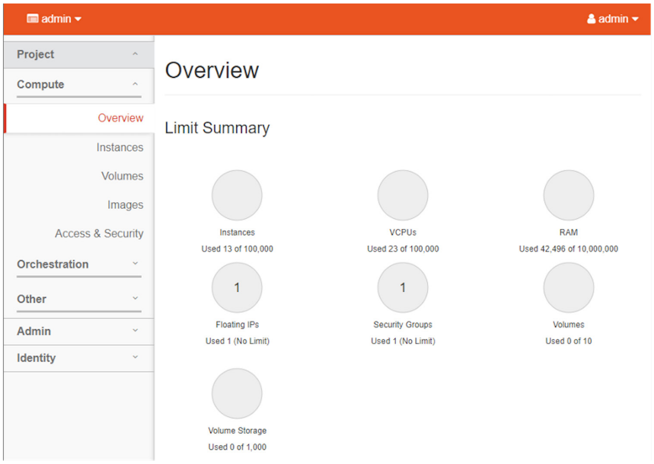


Figure 1.7 *Horizon Dashboard*

Nova

Nova is the project in OpenStack responsible for scheduling and managing the life cycle of VM instances across the compute nodes. It is composed of API, DB, conductor, scheduler, and compute processes:

- The API provides the RESTful interface that can be used by the client or other projects to interact with Nova for compute requirements.
- Database for storing data related to the compute service.
- Conductor handles the requests that will require coordination (build and resize), acts a database proxy, or handles object conversions.
- Scheduler decides the compute server on which a VM can be instantiated.
- Compute services manages communication between the hypervisor and VMs. Except for this service, all other services of Nova will be residing on a controller node. An agent service running on compute will be interacting with compute service on the controller to get the VM into the desired state.

Neutron

The Neutron project networks the devices managed by Nova. The project handles creation and management of virtual networking infrastructure, including virtual networks, switches, subnets, and routers.

It has two primary services: one is installed on a controller, such as a neutron controller or plugins, and the second is installed on a compute server, such as L2 agents that connect vNICs to a virtual switch or L3 agents for routing DHCP.

Neutron also provides firewall as a service, VPN as a service, and load balancer as a service, through plugins (see Figure 1.8).

Swift

Swift in OpenStack is used for storing large amounts of data in a cluster of standardized inexpensive servers instead of specialized storage devices. All objects stored in object storage have a URL, and a RESTful API can be used to interact with the storage system.

Cinder

Cinder is the OpenStack block storage service for providing volumes to Nova VMs, Ironic bare metal hosts, or containers.

Glance

The Glance image services include discovering, registering, and retrieving VM images. Glance has a RESTful API that allows querying of VM image metadata as well as retrieval of the actual image. VM images made available through Glance can be stored in a variety of locations from simple file systems to object storage systems like the OpenStack Swift project.

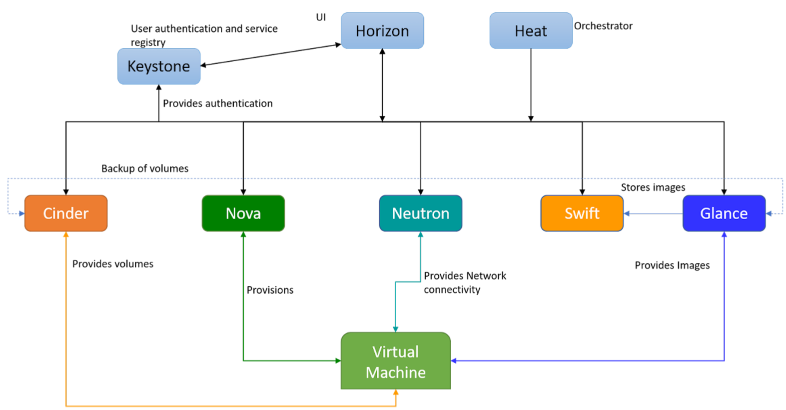


Figure 1.8 Openstack Components Interactions Overview

Understanding Juniper Contrail and Components

Juniper Networks Contrail Networking is a simple, open, and agile SDN solution that automates and orchestrates the creation of highly scalable virtual networks. It seamlessly integrates with several orchestrators such as OpenStack, Kubernetes, OpenShift, Mesos, and VMware.

Contrail is primarily divided into control plane components and a data forwarding element called *vRouter*. The control side is further subdivided into configuration, control, and analytics nodes as you can see in Figure 1.9.

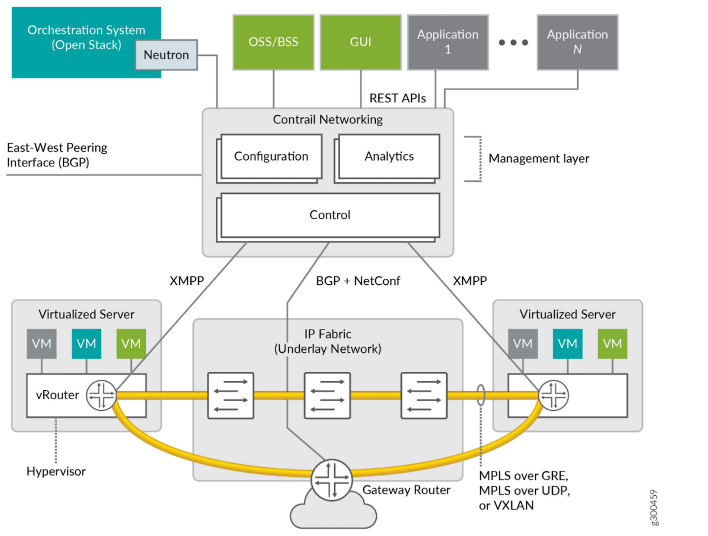


Figure 1.9 Contrail Networking Architecture

Configuration Node

This node is responsible for exposing REST APIs to take high level intents as input (what) and translating them to low level objects (how) that can be pushed down to the forwarding plane through the control node. It is also responsible for storing state of schema objects persistently.

Control Node

The control node implements an in-memory distributed database that contains the transient state of the network such as current network reachability information For each virtual-network defined at the configuration level, there exists one or more routing-instances that contain the corresponding network reachability. A routing-instance contains the IP host routes of VMs in the network, as well as the routing information corresponding to other virtual networks that VMs are allowed to communicate with directly.

Control node processes federate with each other using the BGP protocol, and specifically the BGP/MPLS L3VPN extensions. The same protocol is used to interoperate with physical routers that support network virtualization.

The control node also contains an IF-MAP subsystem, which is responsible for filtering the configuration database and propagating to the compute node agents only the subset of configuration information that is relevant to them.

Communication between the control node and the agent is done via an XMPP-like control channel. While the message format follows the XMPP specification, the state machine does not. Yet. This will be corrected in the near future.

The control node receives subscription messages from the agent for routing information on specific routing-instances and for configuration information pertaining to VMs instantiated in the respective compute node. It then pushes any current information, along with updates, to both routing and configuration objects relevant to each agent. Each agent connects to two control nodes in an active-active model.

All the routing functionality in the system happens at the control node level. The agent is unaware of policies that influence which routes are present in each routing table.

Analytics Node

The analytics node provides a REST interface to a time series database containing the state of various objects in the system such as virtual networks, virtual machines, configuration, and control nodes, as well as traffic flow records. The collected data is stored in a distributed database (Apache Cassandra) for scale out.

The collected objects are defined using an interactive data language (Sandesh) so that the schema of database records can be easily communicated to users or applications performing queries. Sandesh unifies all diagnostics information on the system including data available through the HTTP interface provided by each contrail process.

Information can be collected periodically or based on event triggers.

vRouter or Compute Node

vRouter is the data path component of the Contrail Networking SDN solution. It replaces Linux bridges and OVS on Linux-based computes. It has several advantages over other open-source data path solutions: it can perform routing, L4 security, and bridging all within the same node if both source and destination exist on compute. When a compute node is provisioned as a Contrail vRouter node, the deployment software also installs the required binaries to run vRouter on it.

The compute node is composed of the vRouter *agent* and *data path*. The data path runs as a loadable kernel module on the host operating system or in user space as the DPDK process.

The agent is a user space process that communicates with the control node. It runs a thrift service that is used by the compute node plugin (for example, Nova or XAPI) to enable the virtual interface associated with a VM instance.

The data plane associates each virtual interface with a VRF table and performs the overlay encapsulation functionality. It implements Access Control Lists (ACLs), NAT, multicast replication, and mirroring.

MORE? For detailed explanations of each of these components go to the excellent Contrail documentation in the Juniper TechLibrary: https://www.juniper.net/documentation/en_US/contrail20/topics/concept/understanding-contrail-networking-components.html.

Contrail in OpenStack Environment

In the OpenStack environment, a monolithic plugin of Contrail completely replaces the neutron server. Network related requests from other components of OpenStack are intercepted by this plugin and passed to the Config node of Contrail Networking; see Figure 1.10. Contrail Networking also has its own UI for configuration of objects unique to it.

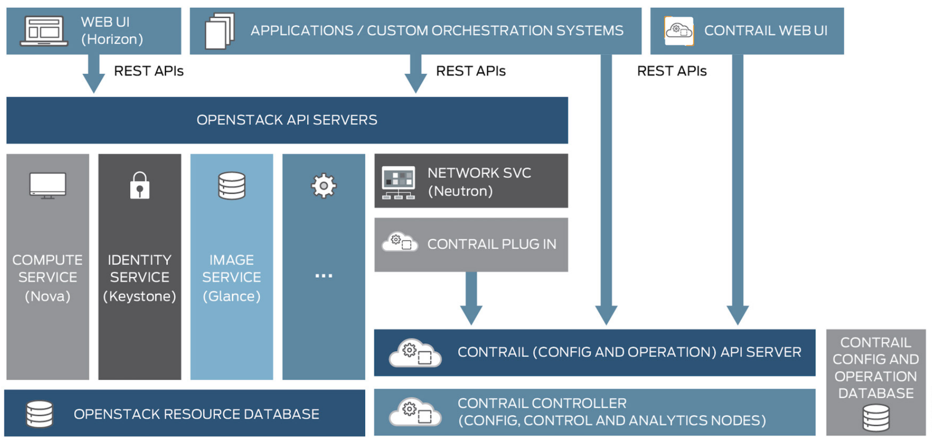


Figure 1.10 Contrail Configuration and OpenStack–API Interaction

Chapter 2

Installation of Kolla-based OpenStack and Contrail Networking

This chapter shows you how to install Kolla-based OpenStack and Contrail in HA on a set of servers that will have control plane and data plane components separated. At this stage, the reader is expected to know how to access the console of the bare metal servers.

Installation Prerequisites

Let's first understand the software and hardware requirements to successfully install OpenStack and Contrail.

You can find the software compatibility matrix in the following link: https://www.juniper.net/documentation/en_US/release-independent/contrail/topics/reference/contrail-supported-platforms.pdf.

In this book, we install Contrail Networking version 2003. Based on the supported platforms document, Contrail version 2003 is supported on CentOS 7.7 with kernel 3.10.0-1062.

Minimum hardware requirements are documented for each version here: https://www.juniper.net/documentation/en_US/contrail20/topics/task/installation/hardware-reqs-vnc.html. Based on the document, each server must have at least 64GB RAM and 300GB of HDD, four CPUs and a minimum of one Ethernet port.

Contrail with OpenStack can also be set up on single node for experimentation or learning purposes. This type of setup is referred to as *all-in-one* or *AIO*. However, in this book we will be installing cluster with one OpenStack node, three Contrail nodes, and two Contrail compute nodes.

OpenStack and Contrail control components can be hosted on VMs. However, the compute nodes have to be bare metal servers to support KVM. Based on this understanding, the test case in this chapter requires:

- X86-64 Servers, three count. One designated as KVM for hosting OpenStack and Contrail control plane components, and two BMS as compute nodes.
- The BMS designated as KVM host should have a minimum of 256 GB RAM and 1TB HDD with 16 CPUs to start with.
- The bare metal servers designated as computes can be made up of any x86-64 servers with a minimum of 128 GB RAM, 512GB HDD, and a number of cores to support virtualization needs.
- One MX Series.
- A switch to connect the servers and the router.
- Two subnets: one for management and other for data plus control traffic.
- Credentials for accessing Contrail Docker private hub, *hub.juniper.net*. Contrail container registry credentials can be requested by emailing *contrail-registry@juniper.net*.

Physical Topology

This chapter's topology is shown in Figure 2.1.

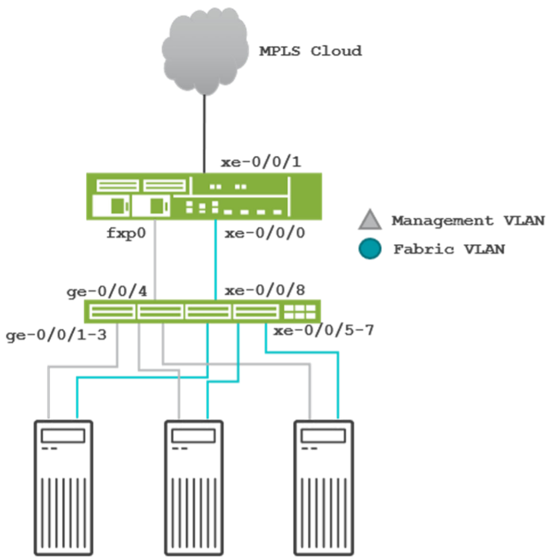


Figure 2.1 Physical Connectivity for This Day One Book

Figure 2.1 illustrates the physical connectivity of all three servers and the MX router to the QFX switch. Each bare metal server has two interfaces connected to the QFX, one for management traffic and the other for fabric interface. In Contrail Networking, the interface that carries the overlay/tenant traffic along with xmpp traffic from vRouter to Contrail controller is called a *fabric interface*.

Logical Topology

Let’s try to understand the logical topology by looking at Figure 2.2. You can see that the first bare metal server is designated as the KVM host. On the KVM host you have to create two bridges, one for management and one for fabric. These bridges will be used to connect the VMs vNICs to the physical world. In the same way, the bare metal servers designated as computes also have two of their interfaces connected to the management VLAN and fabric VLAN, respectively.

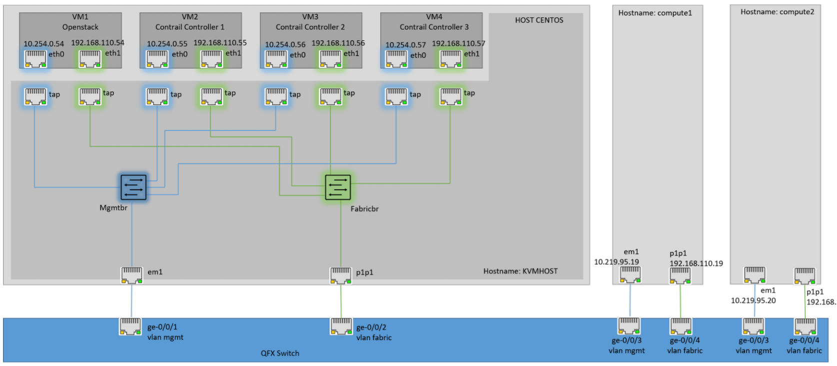


Figure2.2 Logical Topology of the Connectivity of the VMs and Bare Metal Servers

Installing CentOS on Bare Metal Servers

Access the console of the BMS using the browser that is compatible with the out-of-band management (OOM) platform of the BMS. OOM software examples are: iDRAC, iLO, Intel BMC, etc.

For the server designated as the KVM host, use the “virtualization host” option while selecting software for the CentOS installation. Use the option “minimal” when selecting software for the other two servers.

NOTE Make sure that the IP addresses and hostnames for the bare metal servers are as per the logical topology diagram, and verify whether the virtualization extensions on the CPUs are enabled from BIOS:

```
grep -E 'svm|vmx' /proc/cpuinfo
```

If the output looks something like the following, then you can assume that the virtualization extensions are enabled:

```
root@KVM/]$ grep -E 'svm|vmx' /proc/cpuinfo
flags        : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts
acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc art arch_perfmon pebs
bts rep_good nopl xtopology nonstop_tsc aperfmperf eagerfpu pni pclmulqdq dtes64 monitor ds_cpl vmx
smx est tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid dca sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer
aes xsave avx f16c rdrand lahf_lm abm 3dnowprefetch epb cat_l3 cdp_l3 intel_ppin intel_pt ssbd mba
ibrs ibpb stibp ibrs_enhanced tpr_shadow vnmi flexpriority ept vpid fsgsbase tsc_adjust bmi1 hle avx2
smep bmi2 erms invpcid rtm cqm mpx rdt_a avx512f avx512dq rdseed adx smap clflushopt clwb avx512cd
avx512bw avx512vl xsaveopt xsavec xgetbv1 cqm_llc cqm_occup_llc cqm_mbm_total cqm_mbm_local dtherm
ida arat pln pts pku ospke avx512_vnni md_clear spec_ctrl intel_stibp flush_l1d arch_capabilities
```

If not, follow the BIOS/BMS vendor procedure to enable virtualization extensions.

Preparing KVM host for Hosting Control Nodes

Prepare the server designated as KVM host to host the OpenStack controller and Contrail control nodes in high availability. For this we have to create two bridges, which will be used for connecting the VMs to overlay management and data plus control for the controllers and compute nodes.

Once the installation is completed on the KVM host, create two bridges (see Figure 2.2), and map a physical port to extend the connectivity of the VMs to the physical world with the procedure below:

1. Verify that the bridge kernel module on Linux is loaded.

Note that by default, CentOS comes with bridge module loaded:

```
[root@KVMHOST ~]# modinfo bridge
filename:       /lib/modules/3.10.0-1062.el7.x86_64/kernel/net/bridge/bridge.ko.xz
alias:          rtnl-link-bridge
version:        2.3
license:        GPL
retpoline:      Y
rhelversion:    7.7
srcversion:     24DDA8C6E1594CDB8543B49
depends:        stp,llc
intree:         Y
vermagic:       3.10.0-1062.el7.x86_64 SMP mod_unload modversions
signer:         CentOS Linux kernel signing key
sig_key:        51:08:4E:41:88:03:02:BE:5C:B0:74:AC:0D:A3:FE:10:23:3B:7F:1C
sig_hashalgo:   sha256
```

If the bridge module is not loaded, then you can manually load it by running this command:

```
# modprobe --first-time bridge
```

2. As we have selected minimal software selection during installation, we will not have utilities to manage the bridge through the command line of Linux. Hence, let's install bridge utilities:

```
yum install bridge-utils -y

vi /etc/sysconfig/network-scripts/ifcfg-mgmtbr

DEVICE="mgmtbr"
BOOTPROTO="static"
IPADDR="10.253.0.1"
NETMASK="255.255.255.0"
GATEWAY="10.253.0.254"
DNS1=192.168.12.2
ONBOOT="yes"
TYPE="Bridge"
NM_CONTROLLED="no"

vi /etc/sysconfig/network-scripts/ifcfg-em1


DEVICE=em1
TYPE=Ethernet
BOOTPROTO=none
ONBOOT=yes
NM_CONTROLLED=no
BRIDGE=mgmtbr


vi /etc/sysconfig/network-scripts/ifcfg-fabricbr

DEVICE="fabricbr"
BOOTPROTO="static"
IPADDR="192.168.110.1"
NETMASK="255.255.255.0"
ONBOOT="yes"
TYPE="Bridge"
NM_CONTROLLED="no"

vi /etc/sysconfig/network-scripts/ifcfg-p1p1


DEVICE=p1p1
TYPE=Ethernet
BOOTPROTO=none
ONBOOT=yes
NM_CONTROLLED=no
BRIDGE=fabricbr

systemctl restart network
```

MORE? Detailed information about Linux bridges can be found in RedHat documentation, here: https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/networking_guide/sec-network_bridging_using_the_command_line_interface.

Creating a VM for Control Plane Components

Instead of going through the pain of installing Centos on VMs, let's simply use the cloud images that are already in the form of VM disks and customize them for our use.

By default, cloud images do not support password login and there is no default password set for root. To customize the images, use the `virt-customize` command in Linux and set the password for root before importing the image as a VM.

1. Download the Centos Cloud image:

```
[root@KVMHOST ~]# mkdir VMdisks
[root@KVMHOST VMdisks]# cd VMdisks
[root@KVMHOST VMdisks]# wget https://cloud.centos.org/centos/7/images/CentOS-7-x86_64-
GenericCloud-2003.qcow2
```

The downloaded cloud image works well in an environment where the necessary tools are installed that provision the VM with settings such as hostname, IP address, etc. However, in this simple *Day One* setup, don't go to the lengths of installing and configuring these tools. Instead, use the image modification tools to set root password, increase the partition size, and manually configure hostname and other settings.

2. Install the `virt-customize` package. This package allows you to modify the root password of VM image:

```
[root@KVMHOST VMdisks]# yum install /usr/bin/virt-customize
```

3. Set the root password:

```
[root@KVMHOST VMdisks]# virt-customize -a CentOS-7-x86_64-GenericCloud-2003.qcow2 --root-password
password:Juniper
```

4. Check if the disk and partition size of the VM image satisfies the minimum size requirements of Contrail Networking: (https://www.juniper.net/documentation/en_US/release-independent/contrail/topics/reference/contrail-supported-platforms.pdf).

```
[root@KVMHOST VMdisks]# qemu-img info CentOS-7-x86_64-GenericCloud-2003.qcow2
image: CentOS-7-x86_64-GenericCloud-2003.qcow2
file format: qcow2
virtual size: 8.0G (8589934592 bytes)
disk size: 827M
cluster_size: 65536
Format specific information:
    compat: 1.1
    lazy refcounts: false
```

The size of the downloaded cloud image may not suit the requirement, so:

5. Increase the size of the disk to the desired size using the `qemu-img resize` command. This is being done to support the minimum disk size requirements of Contrail listed in the Juniper TechLibrary:

```
[root@KVMHOST VMdisks]# qemu-img resize CentOS-7-x86_64-GenericCloud-2003.qcow2 300G
```

6. Verify the virtual disk size of the image:

```
[root@KVMHOST ~] qemu-img info CentOS-7-x86_64-GenericCloud-2003.qcow2
image: CentOS-7-x86_64-GenericCloud-2003.qcow2
file format: qcow2
virtual size: 300G (322122547200 bytes)
disk size: 896M
cluster_size: 65536
Format specific information:
    compat: 1.1
    lazy refcounts: false
```

Notice the virtual disk size has changed to 300G. However, the partition size is not yet modified. To verify this, let's create a VM with this image and verify the size of the partition:

7. Create a VM with the qcow2 image as in Step 4:

```
[root@KVMHOST VMdisks]# virt-install \
  --name TEMP  \
  --memory 32768 \
  --vcpus 4 \
  --disk /root/VMdisks/CentOS-7-x86_64-GenericCloud-2003.qcow2 \
  --import \
  --os-variant rhel7 \
  --graphics=vnc
```

8. Once the VM is up, log in to the VM using the console:

```
[root@KVMHOST VMdisks]# virsh list
 Id    Name                           State
----------------------------------------------------
 1   TEMP                        running

[root@KVMHOST VMdisks] virsh console 1
Connected to domain TEMP
Escape character is ^]   << press enter here

CentOS Linux 7 (Core)
Kernel 3.10.0-1127.el7.x86_64 on an x86_64

locahost login: root
Password:
[root@localhost ~]#
```

Now you are accessing the shell prompt through the console of the VM.

9. Run the `df -h` command to list the partitions and their size:

```
[root@localhost ~]# df -h
Filesystem      Size  Used Avail Use% Mounted on
devtmpfs        7.8G     0  7.8G   0% /dev
tmpfs           7.8G     0  7.8G   0% /dev/shm
tmpfs           7.8G   17M  7.8G   1% /run
tmpfs           7.8G     0  7.8G   0% /sys/fs/cgroup
/dev/sda1       8.0G  849M  7.2G  11% /
tmpfs           1.6G     0  1.6G   0% /run/user/0
```

Notice the partition size is 8.0G.

10. Verify the virtual disk size using `fdisk -l`:

```
[root@localhost ~]# fdisk -l

Disk /dev/sda: 322.1 GB, 322122547200 bytes, 629145600 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk label type: dos
Disk identifier: 0x000940fd

   Device Boot      Start         End      Blocks   Id  System
/dev/sda1   *        2048   629143295   314570624   83  Linux
```

Though the partition size is 8GB, the virtual disk size is over 300GB.

11. Increase the partition to the maximum disk size using the `xfs_growfs` command:

```
[root@localhost ~]# xfs_growfs /dev/sda1
meta-data=/dev/sda1              isize=512    agcount=4, agsize=524224 blks
         =                       sectsz=512   attr=2, projid32bit=1
         =                       crc=1        finobt=0 spinodes=0
data     =                       bsize=4096   blocks=2096896, imaxpct=25
         =                       sunit=0      swidth=0 blks
naming   =version 2              bsize=4096   ascii-ci=0 ftype=1
log      =internal               bsize=4096   blocks=2560, version=2
         =                       sectsz=512   sunit=0 blks, lazy-count=1
realtime =none                   extsz=4096   blocks=0, rtextents=0
data blocks changed from 2096896 to 78642656
```

12. Verify the partition size after resizing it:

```
[root@localhost ~]# df -h
Filesystem      Size  Used Avail Use% Mounted on
devtmpfs        7.8G     0  7.8G   0% /dev
tmpfs           7.8G     0  7.8G   0% /dev/shm
tmpfs           7.8G   17M  7.8G   1% /run
tmpfs           7.8G     0  7.8G   0% /sys/fs/cgroup
/dev/sda1       300G  855M  300G   1% /
tmpfs           1.6G     0  1.6G   0% /run/user/0
```

Now it reflects the required disk and partition size. Let's shut this VM down and use the modified qcow2 image as the base for the four VMs that will be hosting control plane components.

13. Use the combination `ctrl+]` to exit from the virsh console:

```
[root@KVMHOST VMdisks]#
```

14. Make four copies of the image file with names such as OpenStack.qcow2, ContrailC1.qcow2, ContrailC2.qcow2, and ContrailC1.qcow2:

```
[root@KVMHOST VMdisks]# cp CentOS-7-x86_64-GenericCloud-2003.qcow2  Openstack.qcow2
[root@KVMHOST VMdisks]# cp CentOS-7-x86_64-GenericCloud-2003.qcow2  ContrailC1.qcow2
[root@KVMHOST VMdisks]# cp CentOS-7-x86_64-GenericCloud-2003.qcow2  ContrailC2.qcow2
[root@KVMHOST VMdisks]# cp CentOS-7-x86_64-GenericCloud-2003.qcow2  ContrailC3.qcow2
```

15. Install VMs using the customized Centos cloud images:

```
[root@KVMHOST VMdisks]# virt-install \
  --name Openstack  \
  --memory 32768 \
  --vcpus 4 \
  --disk /root/VMdisks/Openstack.qcow2 \
  --import \
  --os-variant rhel7 \
  --network bridge=mgmtbr \
  --network bridge=fabricbr \
  --graphics=vnc

 [root@KVMHOST VMdisks]# virt-install \
  --name ContrailC1  \
  --memory 32768 \
  --vcpus 4 \
  --disk /root/VMdisks/ContrailC1.qcow2 \
  --import \
  --os-variant rhel7 \
  --network bridge=mgmtbr \
  --network bridge=fabricbr \
  --graphics=vnc

[root@KVMHOST VMdisks]# virt-install \
  --name ContrailC2  \
  --memory 32768 \
  --vcpus 4 \
  --disk /root/VMdisks/ContrailC1.qcow2 \
  --import \
  --os-variant rhel7 \
  --network bridge=mgmtbr \
  --network bridge=fabricbr \
  --graphics=vnc

[root@KVMHOST VMdisks]# virt-install \
  --name ContrailC3  \
  --memory 32768 \
  --vcpus 4 \
  --disk /root/VMdisks/ContrailC1.qcow2 \
  --import \
  --os-variant rhel7 \
  --network bridge=mgmtbr \
  --network bridge=fabricbr \
  --graphics=vnc
```

16. Log in to each VM through console and set hostnames and IP addresses for the VM interfaces as per the logical topology diagram:

```
[root@openstack ~]# cat /etc/hostname
openstack.example.net
[root@openstack ~]#


[root@openstack ~]# cat /etc/sysconfig/network-scripts/ifcfg-eth0
TYPE="Ethernet"
PROXY_METHOD="none"
BROWSER_ONLY="no"
BOOTPROTO="none"
DEFROUTE="yes"
IPV4_FAILURE_FATAL="no"
IPV6INIT="yes"
IPV6_AUTOCONF="yes"
IPV6_DEFROUTE="yes"
IPV6_FAILURE_FATAL="no"
IPV6_ADDR_GEN_MODE="stable-privacy"
NAME="eth0"
UUID="42108fd1-46e4-49b8-a1b7-729f019c2b0f"
DEVICE="eth0"
ONBOOT="yes"
IPADDR="10.254.0.54"
PREFIX="24"
GATEWAY="10.254.0.1"
>>replace the DNS1 Ip address with the working DNS server IP address of your network
DNS1="10.254.0.1"
IPV6_PRIVACY="no"

[root@openstack ~]# cat /etc/sysconfig/network-scripts/ifcfg-eth1
TYPE=Ethernet
PROXY_METHOD=none
BROWSER_ONLY=no
BOOTPROTO=none
DEFROUTE=yes
IPV4_FAILURE_FATAL=no
IPV6INIT=yes
IPV6_AUTOCONF=yes
IPV6_DEFROUTE=yes
IPV6_FAILURE_FATAL=no
IPV6_ADDR_GEN_MODE=stable-privacy
NAME=eth1
UUID=b5dfe96b-65f9-4644-9008-d2e1b819fe35
DEVICE=eth1
ONBOOT=yes
IPADDR=192.168.110.54
PREFIX=24
IPV6_PRIVACY=no
```

Preparing VMs and BMSs

For Contrail and OpenStack components to function properly, it is imperative to have all nodes time-synced. The easiest way to keep all nodes in sync is by using a common time server. Let's install NTP in order to do so.

1. Install NTP and time sync with an NTP server of your choice or the one that is reachable.

2. Generate a SSH key on the VM designated as installer/OpenStack using:

```
ssh-keygen -t rsa
```

3. Copy the newly generated key to other hosts using scp:

```
[root@openstack ~]# ssh-copy-id -i ~/.ssh/id_rsa.pub root@10.254.0.55
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/root/.ssh/id_rsa.pub"
The authenticity of host '10.254.0.55 (10.254.0.55 )' can't be established.
ECDSA key fingerprint is SHA256:eRz5L1yDhTe2L5CfM8VhxWZOtk6RZnNcMg9UP9DUUQU.
ECDSA key fingerprint is MD5:9c:1e:a1:e1:1b:36:92:68:b9:1f:c6:ec:7a:30:dc:49.
Are you sure you want to continue connecting (yes/no)? yes
/usr/bin/ssh-copy-
id: INFO: attempting to log in with the new key(s), to filter out any that are already installed
/usr/bin/ssh-copy-
id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install the new keys
root@10.254.0.55's password:

Number of key(s) added: 1
```

Now try logging in to the machine, with `ssh root@10.254.0.55` and check that only the key(s) you wanted were added:

```
[root@openstack ~]#
```

4. Similarly, copy the SSH key to other VMs and BMSs using step 3.

5. Verify SSH access to each node without password:

```
[root@openstack ~]# ssh root@10.254.0.55
[root@CONTROLLER1]#
```

6. Create a hosts file that can resolve all nodes names with their IP addresses:

```
[root@openstack ~]# cat /etc/hosts
127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
192.168.110.54  openstack.example.net openstack
::1         localhost localhost.localdomain localhost6 localhost6.localdomain6
10.254.0.54  openstack.example.net openstack
10.254.0.55 CONTROLLER1.example.net CONTROLLER1
10.254.0.56 CONTROLLER2.example.net CONTROLLER2
10.254.0.57 CONTROLLER3.example.net CONTROLLER3
10.254.0.19 compute1.example.net compute1
10.254.0.20 compute2.example.net compute2
```

7. To download the Contrail Ansible deployer, access the Juniper support page for Contrail: https://support.juniper.net/support/downloads/?p=contrail.

NOTE You can download this file either to your desktop and then SCP to the first VM, or you can copy the download link from your browser and use tools like wget to download the file directly to the 10.254.0.54 VM itself.

8. Select the Contrail version that you want to download the Ansible deployer to. Navigate to the section of the page where the application tools are mentioned. See Figure 2.3.

9. Click + to expand.

10. Click the tgz file nextt to the Ansible deployer.

X Application Tools 8 File(s)

Description	Release	File Date	Downloads
Ansible Deployer	2003.1	22 Apr 2020	tgz (12.96MB) Checksums

Figure 2.3 Support.juniper.net Contrail Network Download Page

11. Un-tar the Ansible deployer tgz using:

```
tar -xvzf contrail-ansible-deployer-2003.1.40.tgz
```

12. Install Ansible:

```
yum -y install epel-release
yum -y install git ansible-2.7.10
```

13. Install python-pip:

```
yum install -y python-pip
```

14. Run the following commands:

```
yum -y remove PyYAML python-requests
pip install PyYAML requests
```

15. Navigate to contrail-ansible-deployer/config/:

```
cd contrail-ansible-deployer/config/
```

16. Create an instances.yaml file in this folder using your preferred file editor:

```
---
contrail_configuration:
  AUTH_MODE: keystone
  CLOUD_ORCHESTRATOR: openstack
  CONFIG_DATABASE_NODEMGR__DEFAULTS__minimum_diskGB: 2
  CONFIG_NODEMGR__DEFAULTS__minimum_diskGB: 2
  CONTRAIL_VERSION: "2003.1.40"
  CONTROLLER_NODES: "10.254.0.55,10.254.0.56,10.254.0.57"
  CONTROL_NODES: "192.168.110.55,192.168.110.56,192.168.110.57"
  DATABASE_NODEMGR__DEFAULTS__minimum_diskGB: 2
```

```
  ENCAP_PRIORITY: "MPLSoUDP,MPLSoGRE,VXLAN"
  JVM_EXTRA_OPTS: "-Xms1g -Xmx2g"
  KEYSTONE_AUTH_HOST: "192.168.110.200"
  KEYSTONE_AUTH_URL_VERSION: /v3
  METADATA_PROXY_SECRET: contrail123
  RABBITMQ_NODE_PORT: 5673
  VROUTER_GATEWAY: "192.168.110.252"
global_configuration:
  CONTAINER_REGISTRY: hub.juniper.net/contrail
  CONTAINER_REGISTRY_PASSWORD: **********
  CONTAINER_REGISTRY_USERNAME: JNPR-*********
  REGISTRY_PRIVATE_INSECURE: false
instances:
  c1:
    ip: "10.254.0.55"
    provider: bms
    roles:
      analytics:
      analytics_database:
      config:
      config_database:
      control:
      webui:
  c2:
    ip: "10.254.0.56"
    provider: bms
    roles:
      analytics:
      analytics_database:
      config:
      config_database:
      control:
      webui:
  c3:
    ip: "10.254.0.57"
    provider: bms
    roles:
      analytics:
      analytics_database:
      config:
      config_database:
      control:
      webui:
  cp1:
    ip: "10.254.0.19"
    provider: bms
    roles:
      openstack_compute:
      vrouter:
        VROUTER_GATEWAY: "192.168.110.252"
  cp2:
    ip: "10.254.0.20"
    provider: bms
    roles:
      openstack_compute:
      vrouter:
        VROUTER_GATEWAY: "192.168.110.252"
  o1:
```

```
    ip: "10.254.0.54"
    provider: bms
    roles:
      openstack_control:
      openstack_monitoring:
      openstack_network:
      openstack_storage:
kolla_config:
  kolla_globals:
    contrail_api_interface_address: "10.254.0.55 10.254.0.56 10.254.0.57"
    enable_haproxy: true
    enable_ironic: false
    keepalived_virtual_router_id: 55
    kolla_external_vip_address: "10.254.0.54"
    kolla_internal_vip_address: "192.168.110.200"
    openstack_release: queens
  kolla_passwords:
    keystone_admin_password: contrail123
    metadata_secret: contrail123
provider_config:
  bms:
    domainsuffix: example.net
    ntpserver: "ntp.juniper.net"
    ssh_pwd: Juniper
    ssh_user: root
```

Installing Openstack Kolla and Contrail Networking

```
[root@openstack contrail-ansible-deployer]# cd
[root@openstack contrail-ansible-deployer]#  ansible-
playbook -e orchestrator=openstack -i inventory/ playbooks/configure_instances.yml

[WARNING]: Found both group and host with same name: localhost
PLAY [Create container host group]
TASK [Gathering Facts]
ok: [localhost]
TASK [Set orchestrator if not passed]
--snip--
TASK [docker : activate docker login] ***********************************************************
PLAY RECAP ***********************************************************
10.254.0.19              : ok=48   changed=9    unreachable=0    failed=0
10.254.0.20              : ok=48   changed=9    unreachable=0    failed=0
10.254.0.54              : ok=47   changed=23   unreachable=0    failed=0
10.254.0.55              : ok=47   changed=9    unreachable=0    failed=0
10.254.0.56              : ok=47   changed=9    unreachable=0    failed=0
10.254.0.57              : ok=47   changed=9    unreachable=0    failed=0
localhost                : ok=55   changed=8    unreachable=0    failed=0
[root@openstack contrail-ansible-deployer]# ansible-playbook -i inventory/ playbooks/install_
openstack.yml

PLAY [Create container host group for OpenStack]
TASK [Expose instances]
ok: [localhost]
TASK [Expose global_configuration]
--snip--
```

```
10.254.0.19             : ok=69   changed=7    unreachable=0    failed=0
10.254.0.20             : ok=60   changed=5    unreachable=0    failed=0
10.254.0.54             : ok=272  changed=38   unreachable=0    failed=0
10.254.0.55             : ok=3    changed=0    unreachable=0    failed=0
10.254.0.56             : ok=3    changed=0    unreachable=0    failed=0
10.254.0.57             : ok=3    changed=0    unreachable=0    failed=0
localhost               : ok=55   changed=2    unreachable=0    failed=0

[root@openstack contrail-ansible-deployer]#  ansible-
playbook -e orchestrator=openstack -i inventory/ playbooks/install_contrail.yml

PLAY [Create container host group and evaluate variables for Contrail]
TASK [Gathering Facts]
ok: [localhost]
TASK [Expose instances]
ok: [localhost]
TASK [Expose global configuration]
--snip--
10.254.0.19             : ok=21   changed=3    unreachable=0    failed=0
10.254.0.20             : ok=21   changed=3    unreachable=0    failed=0
10.254.0.54             : ok=10   changed=1    unreachable=0    failed=0
10.254.0.55             : ok=65   changed=27   unreachable=0    failed=0
10.254.0.56             : ok=65   changed=27   unreachable=0    failed=0
10.254.0.57             : ok=53   changed=27   unreachable=0    failed=0
localhost               : ok=54   changed=1    unreachable=0    failed=0
```

If you face any difficulty during the above steps, reach out to https://www.juniper.net/documentation/product/en_US/contrail-networking/19/.

IP Address Summary Table

	Interface	IP address
Node	em1	10.254.0.54
	em2	192.168.110.54
CONTROLLER1	em1	10.254.0.55
	em2	192.168.110.55
CONTROLLER2	em1	10.254.0.56
	em2	192.168.110.56
CONTROLLER2	em1	10.254.0.57
	em2	192.168.110.57
Compute1	em1	10.254.0.19
	p1p1	192.168.110.19
Compute2	em1	10.254.0.20
	p1p1	192.168.110.20

Chapter 3

Accessing the Dashboards and CLIs of OpenStack and Contrail Networking

Although operations in cloud environments are performed through APIs, it is still preferable to have access through either the UI or the CLI. As mentioned in Chapter 2, OpenStack comes with a default web UI called Horizon. Apart from this, users can also access the OS through a CLI known as OpenStack Client. It brings command set for Compute, Identity, Image, Object Storage and Block Storage APIs together in a single shell with a uniform command structure.

Accessing OpenStack CLI

In Kolla-based OpenStack installation, the CLI commands can be executed from the Kolla toolbox container provided in the installation. From here you can run the commands to gather information, create, and modify objects in OpenStack.

1. To access the CLI of OpenStack, SSH to the OpenStack node:

```
>  ssh root@10.254.0.54
password:
[root@openstack ~]# docker ps | grep toolbox
bc6c315f939d        kolla/centos-binary-kolla-toolbox:queens               "dumb-
init --single-â€¦"   5 months ago        Up 2 days                              kolla_toolbox
```

2. Gain access to the bash shell of the `kolla-toolbox` container using:

```
[root@openstack ~]# docker exec -it bc6c315f939d bash
(kolla-toolbox)[ansible@openstack /]$
```

3. Run the following command to obtain the list of projects on the OpenStack cluster:

```
(kolla-toolbox)[ansible@openstack /]$ openstack project list
Missing value auth-url required for auth plugin password
(kolla-toolbox)[ansible@openstack /]$
```

You can see that the output suggests we are not passing the user credentials while executing the command. You can pass these parameters through the OpenStack command itself by using the various switches, or you can pass them through environmental variables sourced through a text file. Sourcing through a text file is always easier, since you don't have to key in the lengthy switches and credentials each time you execute an OpenStack command.

By default, the admin credential file is stored under /etc/kolla/kolla-toolbox folder of host file system with the file name as admin-openrc.sh. This file can be accessed within the container bash using the path /var/lib/kolla/config_files.

4. Change the current directory to the one shown below:

```
(kolla-toolbox)[ansible@openstack /]$ cd /var/lib/kolla/config_files/
(kolla-toolbox)[ansible@openstack /var/lib/kolla/config_files]$
```

5. Use the command `source <filename>` to load the environment variable to memory:

```
(kolla-toolbox)[ansible@openstack /var/lib/kolla/config_files]$ source admin-openrc.sh
```

6. Let's try to list the projects again:

```
(kolla-toolbox)[ansible@openstack /var/lib/kolla/config_files]$ openstack project list
+----------------------------------+-------------------------------------------------------------+
| ID                               | Name                                                         |
+----------------------------------+-------------------------------------------------------------+
| 341092a349544c7da62b6794a50b6695 | admin                                                        |
| 4c2cbe28844f4fe69eb2fab0fdac4ed0 | service                                                      |
| d759400944ff40838e78c5b06c3f39a2 | 341092a349544c7da62b6794a50b6695-8867c174-c945-42a8-b8fa.  |
+----------------------------------+-------------------------------------------------------------+
```

By default, OpenStack will be populated with three projects: admin, service, and Heat. We'll be creating a project of our own. For now, let's obtain more information about the project name admin.

7. Get more details of the project named `admin`:

```
(kolla-toolbox)[ansible@openstack /var/lib/kolla/config_files]$ openstack project show admin
+-------------+-----------------------------------------------+
| Field       | Value                                         |
+-------------+-----------------------------------------------+
| description | Bootstrap project for initializing the cloud. |
| domain_id   | default                                       |
| enabled     | True                                          |
| id          | 341092a349544c7da62b6794a50b6695              |
| is_domain   | False                                         |
| name        | admin                                         |
| parent_id   | default                                       |
| tags        | []                                            |
+-------------+-----------------------------------------------+
```

8. Let's also obtain information about the number of computes and their hosts, IP address, etc.:

```
(kolla-toolbox)[ansible@openstack /var/lib/kolla/config_files]$ openstack hypervisor list
+----+---------------------+-----------------+----------------+-------+
| ID | Hypervisor Hostname | Hypervisor Type | Host IP        | State |
+----+---------------------+-----------------+----------------+-------+
|  1 | compute1.example.net | QEMU           | 192.168.110.19 | up    |
|  2 | compute2.example.net | QEMU           | 192.168.110.20 | up    |
+----+---------------------+-----------------+----------------+-------+
```

Try It Yourself Try out this command: `OpenStack hypervisor show 1`.

9. List the networks present in the OpenStack DB:

```
(kolla-toolbox)[ansible@openstack /var/lib/kolla/config_files]$ openstack network list
+--------------------------------------+------------------------------+--------------------------+
| ID                                   | Name                         | Subnets                  |
+--------------------------------------+------------------------------+--------------------------+
| 63b9a291-c534-4734-84dc-b6c5cac47ac4 | internal_vn_ipv6_link_local | 604b2d22-dfa2-11ea-
b0df525400b05250 |
| a796a029-586c-4df5-8762-5cdb40882ecf | ip-
fabric                   |                              |
+--------------------------------------+------------------------------+--------------------------+
```

10. List images in OpenStack:

```
(kolla-toolbox)[ansible@openstack /var/lib/kolla/config_files]$ openstack image list
+--------------------------------------+----------+--------+
| ID                                   | Name     | Status |
+--------------------------------------+----------+--------+
+--------------------------------------+----------+--------+
```

11. List the VM flavors in OpenStack:

```
(kolla-toolbox)[ansible@openstack /var/lib/kolla/config_files]$ openstack flavor list
+----------------------------+-------+-------+------+-----------+-------+-----------+
| ID                         | Name  |  RAM  | Disk | Ephemeral | VCPUs | Is Public |
+----------------------------+-------+-------+------+-----------+-------+-----------+
+----------------------------+-------+-------+------+-----------+-------+-----------+
```

Accessing Contrail Control Nodes Using the CLI

1. Access the shell prompt of the Contrail control node using SSH:

```
[root@CONTROLLER1 ~]#
```

2. Generate the Contrail status report using:

```
[root@CONTROLLER1 ~]# contrail-status
[root@CONTROLLER1 ~]# contrail-status
Pod          Service       Original Name                   Original Version  State    Id            Status
             redis         contrail-external-redis          2003-40          running  14d3274dbdc5  Up 2 hours
analytics    api           contrail-analytics-api           2003-40          running  94fdb33c4c48  Up 2 hours
analytics    collector     contrail-analytics-collector     2003-40          running  ed7b4c7616ea  Up 2 hours
analytics    nodemgr       contrail-nodemgr                 2003-40          running  28e839128847  Up 2 hours
analytics    provisioner   contrail-provisioner             2003-40          running  9a3560c42fd6  Up 2 hours
```

```
config          api           contrail-controller-config-api       2003-40         running  984c696c71c9  Up 2 hours
config          device-manager  contrail-controller-config-devicemgr   2003-40         running  72d7d6123d83  Up 2 hours
config          dnsmasq        contrail-controller-config-dnsmasq    2003-40         running  d14e67258ec3  Up 2 hours
config          nodemgr        contrail-nodemgr                     2003-40         running  d1b706115ea4  Up 2 hours
config          provisioner     contrail-provisioner                 2003-40         running  541fb0b56d55  Up 2 hours
config          schema         contrail-controller-config-schema     2003-40         running  d0876bf3160b  Up 2 hours
config          stats          contrail-controller-config-stats      2003-40         running  879cdbf63d41  Up 2 hours
config          svc-monitor    contrail-controller-config-svcmonitor  2003-40         running  32f9cffef7cc  Up 2 hours
config-database  cassandra     contrail-external-cassandra          2003-40         running  31c44b07642a  Up 2 hours
config-database  nodemgr       ontrail-nodemgr                     2003-40         running  f00c088b4178  Up 2 hours
config-database  provisioner   contrail-provisioner                 2003-40         running  e0ade38a6946  Up 2 hours
config-database  rabbitmq      contrail-external-rabbitmq           2003-40         running  b2724debda3f  Up 2 hours
config-database  zookeeper     contrail-external-zookeeper          2003-40         running  6703bd471710  Up 2 hours
control         control        contrail-controller-control-control   2003-40         running  9eb7c4b0bbdd  Up 2 hours
control         dns           contrail-controller-control-dns       2003-40         running  69bac859f876  Up 2 hours
control         named          contrail-controller-control-named     2003-40         running  1982b8481ffe  Up 2 hours
control         nodemgr        contrail-nodemgr                     2003-40         running  0d5c68f93102  Up 2 hours
control         provisioner     contrail-provisioner                 2003-40         running  8124fcf704d1  Up 2 hours
database        cassandra      contrail-external-cassandra          2003-40         running  178b38439187  Up 2 hours
database        nodemgr        contrail-nodemgr                     2003-40         running  cb4859fac827  Up 2 hours
database        provisioner     contrail-provisioner                 2003-40         running  268b249208da  Up 2 hours
database        query-engine    contrail-analytics-query-engine       2003-40         running  c47debaca6e2  Up 2 hours
webui          job           contrail-controller-webui-job        2003-40         running  2111fa2a4576  Up 2 hours
webui          web           contrail-controller-webui-web        2003-40         running  776b4b094264  Up 2 hours

== Contrail control ==
control: active
nodemgr: active
named: active
dns: active

== Contrail config-database ==
nodemgr: active
zookeeper: active
rabbitmq: active
cassandra: active

== Contrail database ==
nodemgr: active
query-engine: active
cassandra: active

== Contrail analytics ==
nodemgr: active
api: active
collector: active

== Contrail webui ==
web: active
job: active

== Contrail config ==
svc-monitor: active
nodemgr: active
device-manager: active
api: active
schema: backup
```

There is not much you can do in the control nodes of Contrail from the CLI except troubleshoot the processes. Instructions to Contrail are either provided through the REST API on the config node or through Contrail's own web UI.

Accessing vRouter Through the CLI

Let's access the compute nodes shell prompt through SSH:

```
ssh root@10.254.0.19
[root@compute1 ~]#
```

Get the status of the vRouter using the `contrail-status` command:

```
[root@compute1 ~]# contrail-status
Pod      Service      Original Name           Original Version  State    Id            Status
         rsyslogd                             2003-40           running  233e851c74f0  Up 2 hours
vrouter  agent        contrail-vrouter-agent  2003-40           running  0a5e4862832b  Up 2 hours
vrouter  nodemgr      contrail-nodemgr        2003-40           running  c9e2e4b77432  Up 2 hours
vrouter  provisioner  contrail-provisioner    2003-40           running  dbeac6824c92  Up 2 hours

WARNING: container with original name '' have Pod or Service empty. Pod: '' / Service: 'rsyslogd'.
Please pass NODE_TYPE with pod name to container's env

vrouter kernel module is PRESENT
== Contrail vrouter ==
nodemgr: active
agent: active
```

Contrail CLI Commands

Table 3.1 Overview of Contrail CLI Commands

Command	Description
vif	Inspect packet drop counters in the vRouter.
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.

Except for the vif command, all other commands have to be accessed from the vRouter agent container shell.

Let's first explore the output of the vif, and then spend some more time on executing the other commands from the agent container.

1. Access the list of virtual interfaces (vif) of this vRouter using the `vif` command:

```
[root@compute1 ~]# vif --list
Vrouter Interface Table

Flags: P=Policy, X=Cross Connect, S=Service Chain, Mr=Receive Mirror
       Mt=Transmit Mirror, Tc=Transmit Checksum Offload, L3=Layer 3, L2=Layer 2
       D=DHCP, Vp=Vhost Physical, Pr=Promiscuous, Vnt=Native Vlan Tagged
      Mnp=No MAC Proxy, Dpdk=DPDK PMD Interface, Rfl=Receive Filtering Offload, Mon=Interface is 
Monitored
      Uuf=Unknown Unicast Flood, Vof=VLAN insert/strip offload, Df=Drop New Flows, L=MAC Learning 
Enabled
      Proxy=MAC Requests Proxied Always, Er=Etree Root, Mn=Mirror without Vlan Tag, HbsL=HBS Left Intf
      HbsR=HBS Right Intf, Ig=Igmp Trap Enabled

vif0/0      OS: p1p1 (Speed 10000, Duplex 1) NH: 4
            Type:Physical HWaddr:f8:f2:1e:79:5b:d0 IPaddr:0.0.0.0
            Vrf:0 Mcast Vrf:65535 Flags:L3L2VpEr QOS:-1 Ref:11
            RX packets:37445140  bytes:11457103321 errors:0
            TX packets:21495394  bytes:7233581024 errors:0
            Drops:17

vif0/1      OS: vhost0 NH: 5
            Type:Host HWaddr:f8:f2:1e:79:5b:d0 IPaddr:192.168.110.19
            Vrf:0 Mcast Vrf:65535 Flags:L3DEr QOS:-1 Ref:8
            RX packets:20775115  bytes:7186436748 errors:0
            TX packets:37776153  bytes:11454598926 errors:0
            Drops:18

vif0/2      OS: pkt0
            Type:Agent HWaddr:00:00:5e:00:01:00 IPaddr:0.0.0.0
            Vrf:65535 Mcast Vrf:65535 Flags:L3Er QOS:-1 Ref:3
            RX packets:6900331  bytes:593442958 errors:4
            TX packets:17718654  bytes:1739016720 errors:0
            Drops:4

vif0/4350   OS: pkt3
            Type:Stats HWaddr:00:00:00:00:00:00 IPaddr:0.0.0.0
            Vrf:65535 Mcast Vrf:65535 Flags:L3L2 QOS:0 Ref:1
            RX packets:112  bytes:10976 errors:0
            TX packets:112  bytes:9408 errors:0
            Drops:0

vif0/4351   OS: pkt1
            Type:Stats HWaddr:00:00:00:00:00:00 IPaddr:0.0.0.0
            Vrf:65535 Mcast Vrf:65535 Flags:L3L2 QOS:0 Ref:1
            RX packets:0  bytes:0 errors:0
            TX packets:0  bytes:0 errors:0
            Drops:0
```

Let's understand the output of the default interface, which exists even when a VM is not spun up on a compute (see Table 3.2). The default interfaces are vif0/0, vif0/1, vif0/2, vif0/4350, and vif0/4351. All other interfaces starting from vif0/3 are created upon VM spin up.

Table 3.2 VIF Fields and Their Descriptions

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	Type:Virtual HWaddr:00:00:5e:00:01:00 IPaddr:0
	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: • 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	Vrf:65535 Flags:L3 MTU:1514 Ref:2
	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). Flag options identify that the following are enabled for the interface: • P - Policy. All traffic that comes to vRouter from this interface are subjected to 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. All packets that egresses this interface are mirrored. • Mr - Mirroring receive direction. All packets that ingresses this interface will be mirrored. • Tc - Checksum offload on the transmit side. Valid only on the physical interface
Rx	RX packets:143 bytes:6606 errors:0 Packets received by vRouter from this interface.
TX	TX packets:270 bytes:11924 errors:0 Packets transmitted out by vRouter on this interface

So vif0/0 is always part of vrf0 and will not be associated with an IP address like a vif interface would be. Any packet arriving on the physical interface, p1p1 in this example, will be first processed by the vRouter to check if the packet is encapsulated with either MPLSoUDP, MPLSoGRE, or VxLAN. If so, then it will be processed based on the label or VNI that it carries. If there is no encapsulation, then the packet is forwarded to the vhost0 tap interface towards the host.

Vif0/1 is mapped to vhost0. This interface is used by vRouter to receive and forward traffic to/from compute host OS.

Vif0/2 is mapped to pkt0, which is used to communicate between vRouter agent and vRouter itself.

Vif0/4350 and vif0/4351 are used to receive side steering (RSS) of packets by vRouter to different cores of the CPU.

MORE? More information about vif interfaces and how they function can be found at the Juniper TechLibrary: https://www.juniper.net/documentation/en_US/contrail20/topics/task/configuration/vrouter-cli-utilities-vnc.html.

Running vRouter Commands From Container Shell

To execute vRouter commands on a containerized Contrail compute:

1. Run the `docker ps` command and identify the ID of container named `contrail-vroute-agent:xxxxx`:

```
[root@compute2 ~]# docker ps
CONTAINER ID        IMAGE                                                                COMMAND                CREATED
STATUS              PORTS               NAMES
711f9aac1834        hub.juniper.net/contrail/contrail-vrouter-agent:2003.1.40      "/entrypoint.sh /
usrâ€¦"   2 months ago        Up 2 hours                              vrouter_vrouter-agent_1
59c5f60fcf90        hub.juniper.net/contrail/contrail-nodemgr:2003.1.40            "/entrypoint.sh /
binâ€¦"   2 months ago        Up 2 hours                              vrouter_nodemgr_1
07a7f5a51d44        hub.juniper.net/contrail/contrail-provisioner:2003.1.40        "/entrypoint.sh /
usrâ€¦"   2 months ago        Up 2 hours                              vrouter_provisioner_1
b3a9db1c428e        hub.juniper.net/contrail/contrail-external-rsyslogd:2003.1.40   "/contrail-
entrypoinâ€¦"   2 months ago        Up 2 hours                              rsyslogd_rsyslogd_1
26343be2cb16        kolla/centos-binary-nova-compute:queens                          "dumb-init
--single-â€¦"   2 months ago        Up 2 hours                              nova_compute
5cf2f32e8ce8        kolla/centos-binary-nova-libvirt:queens                          "dumb-init
--single-â€¦"   2 months ago        Up 2 hours                              nova_libvirt
848f7f67a71a        kolla/centos-binary-nova-ssh:queens                              "dumb-init --single-â€¦"
2 months ago        Up 2 hours                              nova_ssh
ccbbe0bd9ddf        kolla/centos-binary-cron:queens                                  "dumb-init --single-â€¦"
2 months ago        Up 2 hours                              cron
8c383a2d59a6        kolla/centos-binary-kolla-toolbox:queens                         "dumb-init
--single-â€¦"   2 months ago        Up 2 hours                              kolla_toolbox
f23c9dfacbb9        kolla/centos-binary-fluentd:queens                               "dumb-init --single-â€¦"
2 months ago        Up 2 hours                              fluentd
[root@compute2 ~]#
```

2. Access the shell of the container by running the `docker exec -it <contrail id> /bin/bash` command.

NOTE Sometimes the prompt may wrap the commands you are typing. To prevent this, we can add `-e COLUMNS=$COLUMNS -e LINES=$LINES` to export the host column and line variable to the container prompt:

```
[root@compute2 ~]# docker exec -ti -e COLUMNS=$COLUMNS -e LINES=$LINES 711f9aac1834 /bin/bash
(vrouter-agent)[root@compute1 /]$
```

Try It Yourself Try running these various commands of vRouter to better understand their syntax: `nh`, `rt`, `mpls`, `dropstats`, `flow`, `vrfstats`, `vxlan`. Here's an example containing a few commands:

```
(vrouter-agent)[root@compute1 /]$ flow -s

2021-01-21 13:17:49 +0530
Flow Statistics
---------------
    Total  Entries  --- Total =       8, new =       8
    Active Entries  --- Total =       8, new =       8
    Hold   Entries  --- Total =       0, new =       0
    Fwd flow Entries  - Total =       6
    drop flow Entries - Total =       0
    NAT flow Entries  - Total =       2

    Rate of change of Active Entries
    --------------------------------
        current rate      =       0
        Avg setup rate    =       0
        Avg teardown rate =       0
    Rate of change of Flow Entries
    ------------------------------
        current rate      =       0

(vrouter-agent)[root@compute1 /]$ flow -l
Flow table(size 161218560, entries 629760)

Entries: Created 111 Added 111 Deleted 116 Changed 132Processed 111 Used Overflow entries 0
(Created Flows/CPU: 1 0 1 1 2 1 2 5 1 1 2 0 7 2 9 0 5 3 4 1 2 1 1 1 2 1 0 7 5 0 3 0 1 0 1 0 0 0 1 18 1 8 0 0
0 0 0 0 2 0 7 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0)(oflows 0)

Action:F=Forward, D=Drop N=NAT(S=SNAT, D=DNAT, Ps=SPAT, Pd=DPAT, L=Link Local Port)
 Other:K(nh)=Key_Nexthop, S(nh)=RPF_Nexthop
 Flags:E=Evicted, Ec=Evict Candidate, N=New Flow, M=Modified Dm=Delete Marked
TCP(r=reverse):S=SYN, F=FIN, R=RST, C=HalfClose, E=Established, D=Dead

    Index                Source:Port/Destination:Port                      Proto(V)
-----------------------------------------------------------------------------------
(vrouter-agent)[root@compute1 /]$
(vrouter-agent)[root@compute1 /]$ nh --list
Id:0          Type:Drop           Fmly: AF_INET  Rid:0  Ref_cnt:55801      Vrf:0
              Flags:Valid,

Id:1          Type:Drop           Fmly: AF_INET  Rid:0  Ref_cnt:2827       Vrf:0
              Flags:Valid, Etree Root,
```

```
Id:3          Type:L2 Receive    Fmly: AF_INET  Rid:0  Ref_cnt:14        Vrf:0
              Flags:Valid, Etree Root,

Id:4          Type:Encap         Fmly: AF_INET  Rid:0  Ref_cnt:1         Vrf:0
              Flags:Valid, Etree Root,
              EncapFmly:0806 Oif:0 Len:14
              Encap Data: f8 f2 1e 79 5b d0 f8 f2 1e 79 5b d0 08 00

Id:5          Type:Encap         Fmly: AF_INET  Rid:0  Ref_cnt:3         Vrf:0
              Flags:Valid, Policy, Etree Root,
              EncapFmly:0806 Oif:1 Len:14
              Encap Data: f8 f2 1e 79 5b d0 f8 f2 1e 79 5b d0 08 00

Id:6          Type:Encap         Fmly: AF_INET  Rid:0  Ref_cnt:3         Vrf:0
              Flags:Valid, Etree Root,
              EncapFmly:0806 Oif:1 Len:14
              Encap Data: f8 f2 1e 79 5b d0 f8 f2 1e 79 5b d0 08 00
--snip--

(vrouter-agent)[root@compute1 /]$ nh --get 1
Id:1          Type:Drop          Fmly: AF_INET  Rid:0  Ref_cnt:2827      Vrf:0
              Flags:Valid, Etree Root,

(vrouter-agent)[root@compute1 /]$ nh --get 3
Id:3          Type:L2 Receive    Fmly: AF_INET  Rid:0  Ref_cnt:14        Vrf:0
              Flags:Valid, Etree Root,

(vrouter-agent)[root@compute1 /]$ nh --get 4
Id:4          Type:Encap         Fmly: AF_INET  Rid:0  Ref_cnt:1         Vrf:0
              Flags:Valid, Etree Root,
              EncapFmly:0806 Oif:0 Len:14
              Encap Data: f8 f2 1e 79 5b d0 f8 f2 1e 79 5b d0 08 00
```

We will explore more of the vRouter command line when we have a few VMs spun up and can understand the life of the packet with the help of these commands.

Accessing OpenStack GUI

Access the Horizon GUI using a browser.

1. Since we are accessing the node through the MGMT network, enter the MGMT IP address of the OpenStack Node as in Figure 3.1.

Figure 3.1 Logging in to OpenStack GUI (Horizon)

2. Enter the username and password and click connect as shown in Figure 3.2.

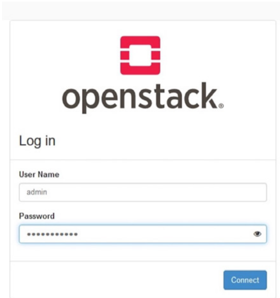


Figure 3.2 OpenStack Log In Page

By default, a first-time login will land you on the Project page of Horizon. Let's take a moment to understand the layout.

The top ribbon has the current project ID and current username

On the left-hand side are the Project, Admin, and Identity tabs:

- The Project tab can be used by users to list out the API endpoints and to manage instances and networks that perform Heat orchestration tasks.
- The Admin tab can be used by administrator users to manage compute, network, and system-related configurations.
- The Identity tab is used to manage projects and users and to assign users to one or more projects as an administrator or as a simple user.

MORE? Details about the OpenStack GUI can be found here: https://docs.openstack.org/horizon/latest/user/log-in.html.

Accessing Contrail GUI

To access the Contrail GUI use the following procedure.

1. Enter the URL https://<contrail_GUI_IP_address>:8143 as shown in Figure 3.3.

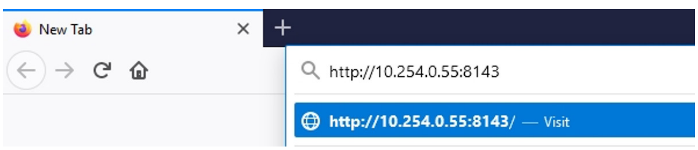


Figure 3.3 Contrail Login URL

2. Enter the username and password as shown in Figure 3.4, and click Sign in.

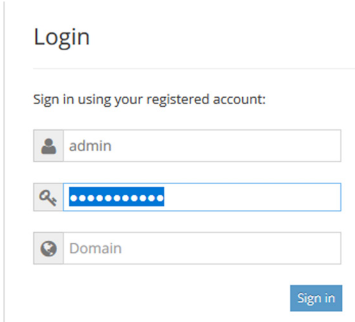


Figure 3.4 Contrail Login Page

Exploring the Contrail GUI

In Figure 3.5 you can see that the Contrail GUI has four main parts:

1. Sidebar with options to switch between monitor, configuration, settings, or query mode.
2. Options for each item mentioned in number 1.
3. Action or content section.
4. User and project selection section.

Let's review the four modes: monitor, configuration, settings, and query.

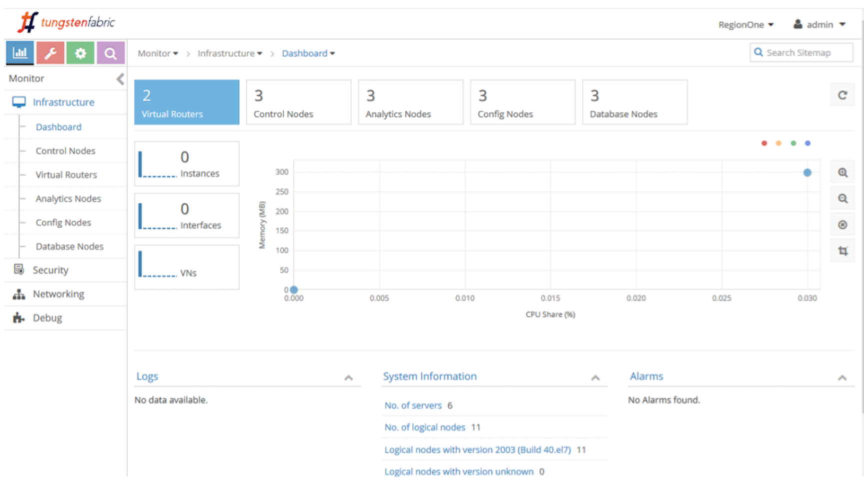


Figure 3.5 Contrail GUI

Monitor

Monitoring options for the infrastructure of Contrail Controller nodes and compute nodes can be found under this tab (see Figure 3.6):

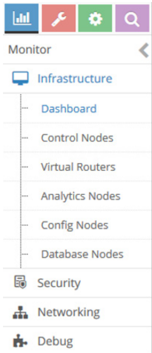


Figure 3.6 Monitor Options in Contrail GUI

Configure

This tab (see Figure 3.7) provides options to configure virtual networks, polices, peer with external BGP routers, IPAM, and add physical devices that can be managed through Contrail, etc.

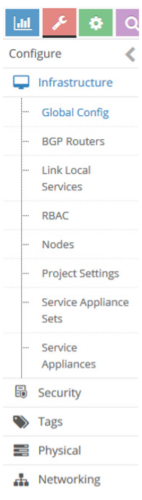


Figure 3.7 Options Under Configuration Tab

Settings

The Settings tab (See Figure 3.8) allows you to explore the Config DB objects and read the contents of each object. It provides access to the Introspect of all nodes in the cluster, and lets you configure Contrail using JSON.

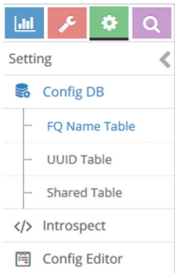


Figure 3.8 Options Under Settings Menu

Note that the menu option Introspect is a service provided in Contrail to gather parameters from various components. It is similar to showing commands in Junos.

Query

This tab (See Figure 3.9) provides options to query and explore the data collected by Analytics nodes, such as Flows, Sessions, Logs, and Statistics.

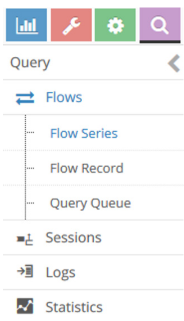


Figure 3.9 Query Menu

The content section of the page is context driven and changes with the tab and subtab you have selected. By default, the Monitoring tab and Dashboard subtab are selected. In the dashboard screen (see Figure 3.10), we can see the status and number of nodes. You can see that this cluster has two compute nodes, three control nodes, three analytics nodes, and three database nodes.

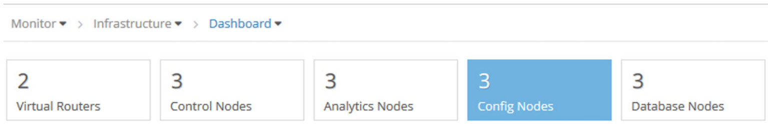


Figure 3.10 Dashboard Subtab

Along with the above information, you can also monitor the number of instances, interfaces, and VNs that are active on the cluster. Figure 3.11 shows the CPU share of the total utilization along with memory utilization on a node.

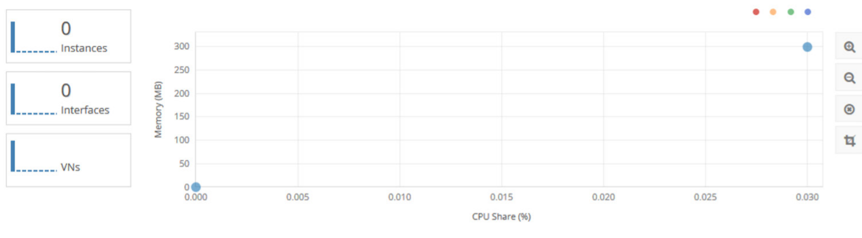


Figure 3.11 CPU Share of the Total Utilization

The bottom section of the dashboard (Figure 3.12) provides the number of servers, number of logical nodes, and the version of each these running nodes.

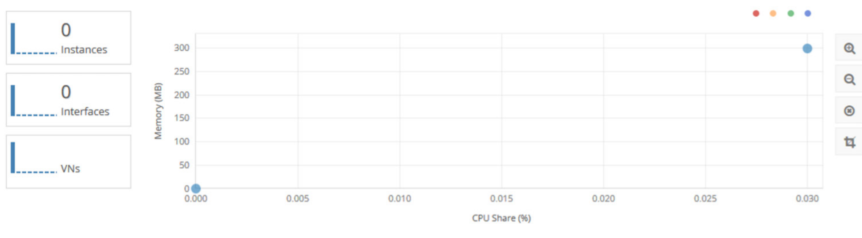


Figure 3.12 Bottom Section of Dashboard

Chapter 4

Setting Up a Three-tier Application

This chapter shows you how to use the OpenStack and Contrail Networking GUI to create projects, user(s), VM flavors, images in OpenStack, virtual network(s), VMs, policies, security groups, and service chains, as well as how to configure BGP-as-a-service to realize a three-tier web application.

Three-tier Web Application

Any properly implemented web application will be divided into three tiers:

- The UI/presentation/web layer (in this layer load-balancers and web server are instantiated).
- The application layer where the business logic (app servers) would reside.
- And the last tier that stores the data (DB servers) to be accessed or generated by the application tier.

Though all these layers could exist in the network, for network security purposes they are usually placed across the firewall/IPS appliance.

Understanding the Three-tier Web Application Topology

Let's spend some time with the topology that we will build in this guide, and understanding the purpose of each virtual network, VM instances, and more.

In Figure 4.1, users from the corporate office or from remote offices that have MPLS connectivity will access the web-based application through L3VPN until the GWR and then use MPLSoGRE to reach the Intranet VN. Source and destination IP subnets are routable end-to-end.

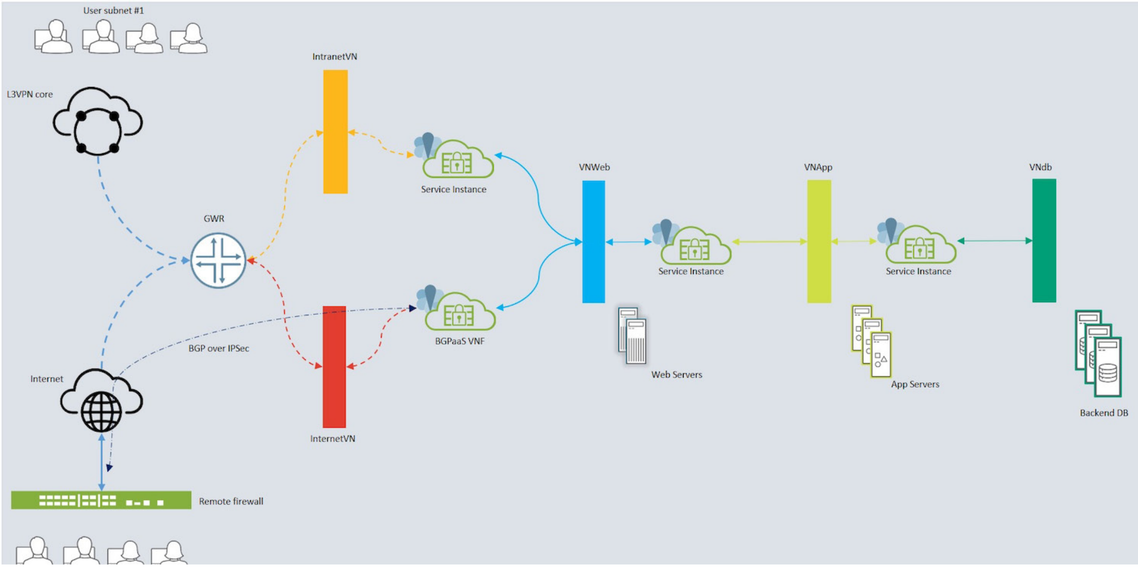


Figure 4.1 Chapter Four's Topology

Users who do not have MPLS-based connectivity will use IPsec tunnels terminated on firewalls at both ends. Source and destination networks are only visible to the IPsec endpoints or any other device besides these firewalls. VNWeb and Usersubnet2 are exchanged through BGPoIPsec, preventing Internet VN and GWR from learning them. User requests originating from usersubnet2 will be encrypted at the remote firewall, and source and destination IP addresses will be the internet routable firewall IPs.

VNWeb will host load balancers and web servers that serve the user with application web pages (front end). This VN will be advertised to the DC gateway by configuring it with an import/export RT matching on Contrail and MX Series router.

VNApp will host the AppServers, which are the backend to the webserver.

DBVn will host database servers that provide data storage and retrieval for the AppServers.

vSRX instances provide security between the multiple tiers. This is explained in more detail in the *Service Chaining* section of this chapter.

One instance of vSRX is configured to peer with the controller to exchange routes with each other using BGP. This is also referred to as BGPaaS. BGPaaS is explained in more detail later in this chapter.

The MX Series router is used as the DC Gateway providing connectivity to MPLS and Internet service providers.

Project in OpenStack

Projects are organizational units in the cloud to which you can assign users. Each project can be defined with its users, virtual networks, images, VMs, etc. You can also configure quota to limit the amount of resource a project can consume from the pooled resources of the cloud.

You can add, update, and delete projects and users, assign users to one or more projects, and change or remove the assignment. To enable or temporarily disable a project or user, update that project or user. You can also change quotas at the project level.

Tenant/Project and User Creation

First, let's create a new tenant/project. Then we'll create a user and map the user to the new project.

Creating a Project

Go to Identity/Projects in the OpenStack GUI, as shown in Figure 4.2.

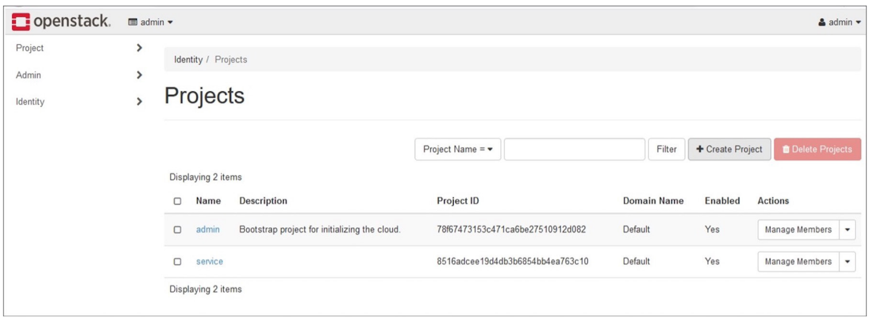


Figure 4.2 Identity > Projects

1. Click on the Identity tab to expand it.
2. Click on projects link to load the projects actions screen.
3. Click on the + Create Project button. Figure 4.3 appears.
4. Enter the project name, such as Dayone, and click on Project members.

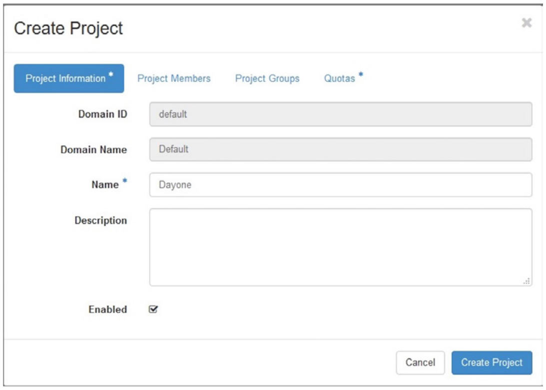


Figure 4.3 Create Project Screen

5. Click on the + button next to the admin to include admin as a user in the Dayone project (Figure 4.4).

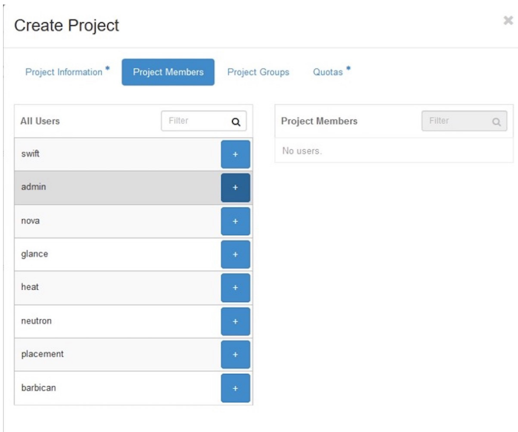


Figure 4.4 Create Project > Project Members

6. Click on drop down list next the user admin in the Project Members section (Figure 4.5) and select admin as the role for the user admin.

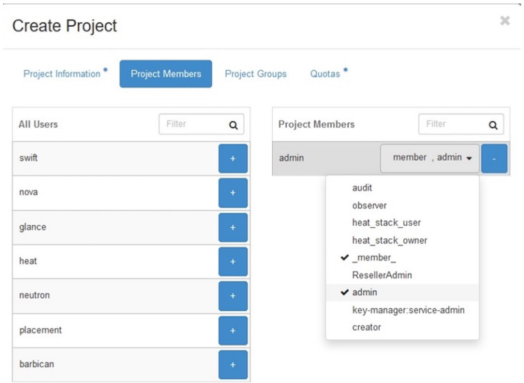


Figure 4.5 Create Project > Project members > Assigning Roles

7. Click on create project (Figure 4.6).

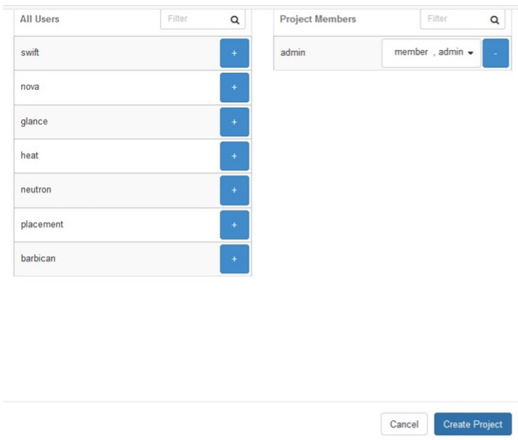


Figure 4.6 Project Members After Adding Role "admin" to User

Create a User and Assign to a Project

1. Click on Users under the Identity tab to load user's actions screen (Figure 4.7).
2. Click on the + Create User button and Figure 4.8 will appear.

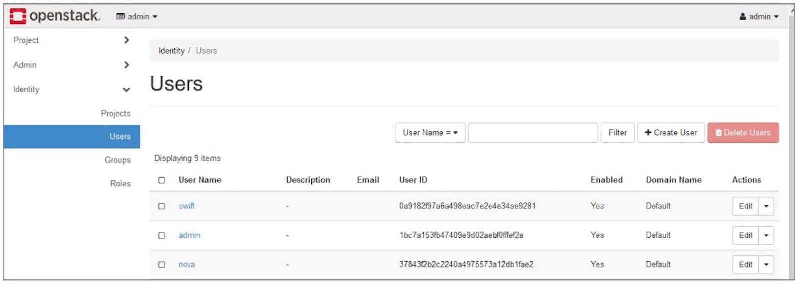


Figure 4.7 Identity > Users

3. Enter the required details such as, User name, password, confirm password.

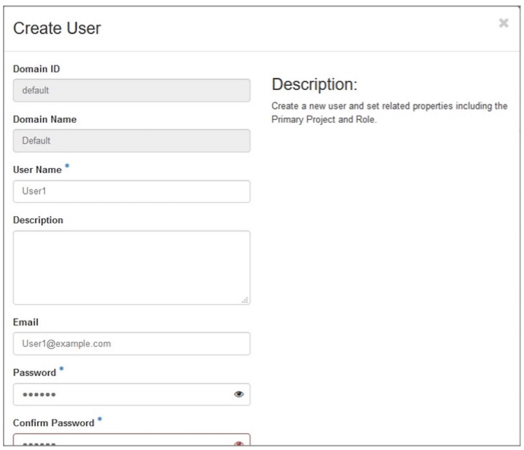


Figure 4.8 User Creation Screen

4. Scroll and select the primary project for the user as Dayone in Figure 4.9.

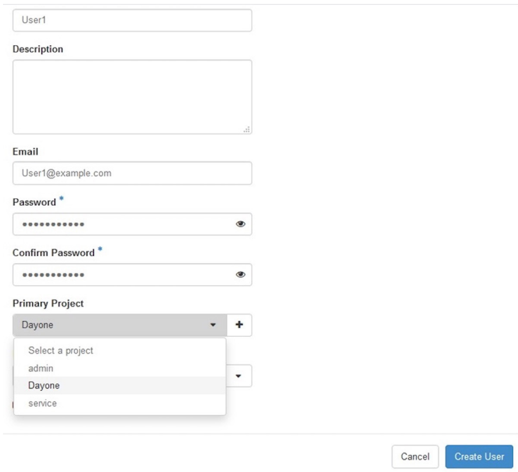


Figure 4.9 *Creating User > Assigning Primary Project*

5. Scroll and select Role as admin in Figure 4.10.

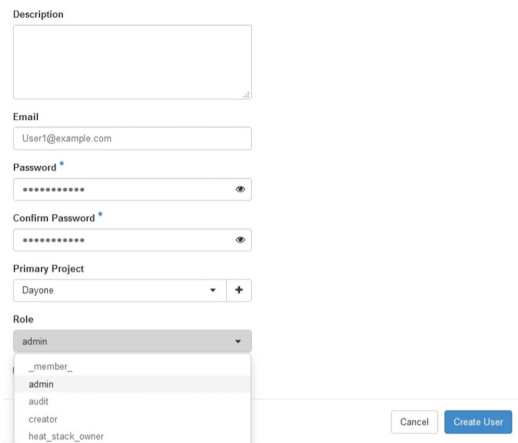


Figure 4.10 *Creating User > Assigning Role to User*

6. Click on the Create User button (Figure 4.11).

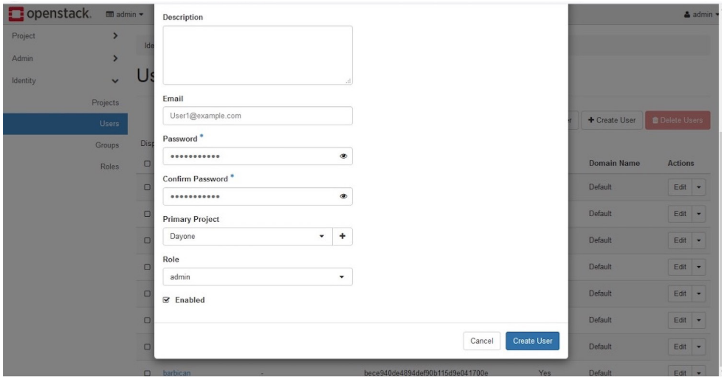


Figure 4.11 Completing User Creation

7. The Users main screen will list your new user (Figure 4.12).

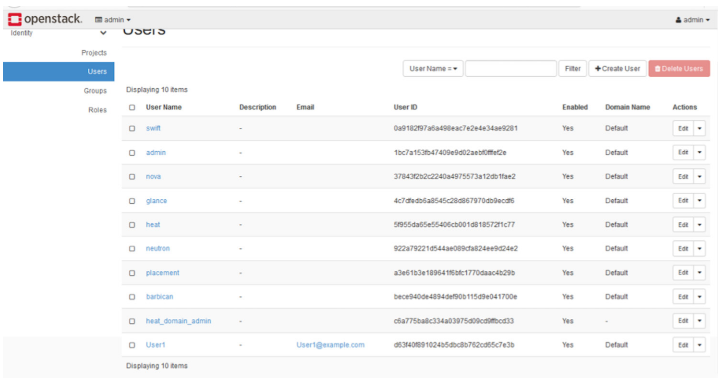


Figure 4.12 User List After Adding User1

Working with OpenStack CLI

Log in to the OpenStack node with root credentials.

When working with the command line you provide the credentials such as project name, domain, tenant, username, password, and auth_url, which you can either provide in each command or set as environment variables. Setting up environment variables is the easiest and quickest way to do this.

By default, admin-openrc.sh is created in the /etc/kolla/kolla-toolbox folder of the OpenStack host node. Here are the contents of the file:

```
cd /etc/kolla/kolla-toolbox
cat admin-openrc.sh
export OS_PROJECT_DOMAIN_NAME=Default
export OS_USER_DOMAIN_NAME=Default
export OS_PROJECT_NAME=admin
export OS_TENANT_NAME=admin
export OS_USERNAME=admin
export OS_PASSWORD=contrail123
export OS_AUTH_URL=http://192.168.10.200:35357/v3
export OS_INTERFACE=internal
export OS_IDENTITY_API_VERSION=3
export OS_REGION_NAME=RegionOne
export OS_AUTH_PLUGIN=password
export OS_BAREMETAL_API_VERSION=1.29
```

1. Let's make a copy of this file and change the tenant, username, and password:

```
[root@OPENSTACK kolla-toolbox]# cp admin-openrc.sh user1-openrc.sh
[root@OPENSTACK kolla-toolbox]# sed -i s/PROJECT_NAME=admin/PROJECT_NAME=Dayone/g user1-openrc.sh
[root@OPENSTACK kolla-toolbox]# sed -i s/TENANT_NAME=admin/TENANT_NAME=Dayone/g user1-openrc.sh
[root@OPENSTACK kolla-toolbox]# sed -i s/USERNAME=admin/USERNAME=User1/g user1-openrc.sh
[root@OPENSTACK kolla-toolbox]# sed -i s/PASSWORD=contrail123/PASSWORD=User1pass/g user1-openrc.sh
```

Now there are two RC files, one for Admin and other one for User1:

```
[root@OPENSTACK kolla-toolbox]# ls -lah | grep rc
-rw-r--r--.  1 root root  398 May 29 12:24 admin-openrc.sh
-rw-r--r--.  1 root root  400 Jun  4 14:07 user1-openrc.sh
```

To execute OpenStack commands in the Kolla environment, you must use the shell of the kolla-toolbox container.

2. To access the shell of the kolla-toolbox container, run the command:

```
docker -it exec kolla-toolbox bash

[root@OPENSTACK kolla-toolbox]# #this is host machine's shell
[root@OPENSTACK kolla-toolbox]# docker exec -it kolla_toolbox bash
(kolla-toolbox)[ansible@OPENSTACK /]$ #this is kolla-toolbox container shell
```

3. Navigate to the kolla/config_files/ folder, which is shared between the host and the container. It can be found under the /var/lib folder of the container:

```
(kolla-toolbox)[ansible@OPENSTACK /]$ cd /var/lib/kolla/config_files/
```

4. Source the file:

```
(kolla-toolbox)[ansible@OPENSTACK /var/lib/kolla/config_files]$ source user1-openrc.sh
```

Verify the credentials by running any OpenStack command.

5. List the Projects in OpenStack database:

```
(kolla-toolbox)[ansible@openstack /]$ openstack project list
+----------------------------------+---------------------------------------------+
| ID                               | Name                                        |
+----------------------------------+---------------------------------------------+
| 341092a349544c7da62b6794a50b6695 | admin                                       |
| 4c2cbe28844f4fe69eb2fab0fdac4ed0 | service                                     |
| 5fd40070eda6437c82b4be7849d33528 | Dayone                                      |
| d759400944ff40838e78c5b06c3f39a2 | 341092a349544c7da62b6794a50b6695-8867c174-
c945-42a8-b8fa-e516809 |
+----------------------------------+---------------------------------------------+
```

6. List the users in OpenStack database:

```
(kolla-toolbox)[ansible@openstack /]$ openstack user list
+----------------------------------+-------------------+
| ID                               | Name              |
+----------------------------------+-------------------+
| 16d6a0ff512d4cddb0f3f62f15780346 | heat              |
| 6e7845f478984650837231361f1511ba | nova              |
| 75d3084b407a4e86a22699b3e02042e3 | neutron           |
| 90a447ba2e2b49aa9d0ecb11be288741 | barbican          |
| 93a4b64e6e484603a63695d28912c91c | placement         |
| b2fa4b8a9fae405e87d554f413727b50 | glance            |
| e80f40773d2c434aa61b82ea6254eedb | heat_domain_admin |
| ecbda7c5ce2045769410e4f609b7cd7d | swift             |
| f57439fddf5b45ca91fdbe3ccb458c03 | User1             |
| fba42103264b4da0a6b11a19f4268143 | admin             |
+----------------------------------+-------------------+
```

7. Show more details of project Dayone:

```
(kolla-toolbox)[ansible@openstack /]$ openstack project show Dayone
+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description |                                  |
| domain_id   | default                          |
| enabled     | True                             |
| id          | 5fd40070eda6437c82b4be7849d33528 |
| is_domain   | False                            |
| name        | Dayone                           |
| parent_id   | default                          |
| tags        | []                               |
+-------------+----------------------------------+
```

The next few sections of this chapter make use of both the OpenStack CLI and the GUI, so be sure you have familiarity with both. Now that a project called Dayone and a user is created and mapped with the project, let's continue to create more objects such as Flavors, Images, VNs, VMs, and more.

Flavors

In OpenStack, *flavors* define the compute, memory, and storage capacity of Nova computing instances. To put it simply, a flavor is an available hardware configuration for a server. It defines the *size* of a virtual server that can be launched.

In earlier versions of OpenStack like Mitaka, the Nova was populated by default with several flavors that could be used to spawn VMs. Post Mitaka, one is expected to create at least one flavor before launching or spawning a VM.

Creating Flavors

1. Expand the Admin tab on left hand side of the dashboard.
2. Expand compute.
3. Click on Flavors.
4. Click the +Create Flavor button (Figure 4.13).

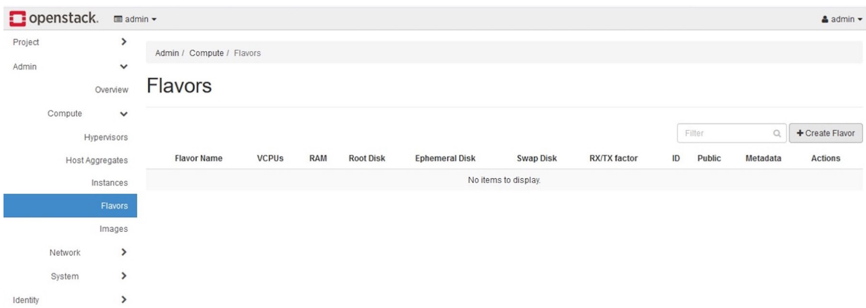


Figure 4.13 Creating Flavor

Enter the following information as shown in Figure 4.14:

- Name: tiny
- ID: leave as default value
- VCPUs: 1
- RAM(MB): 512
- Root Disk(GB): 1
- Ephemeral Disk(GB): 0
- Swap Disk(MB): 0
- RX/TX Factor: 1

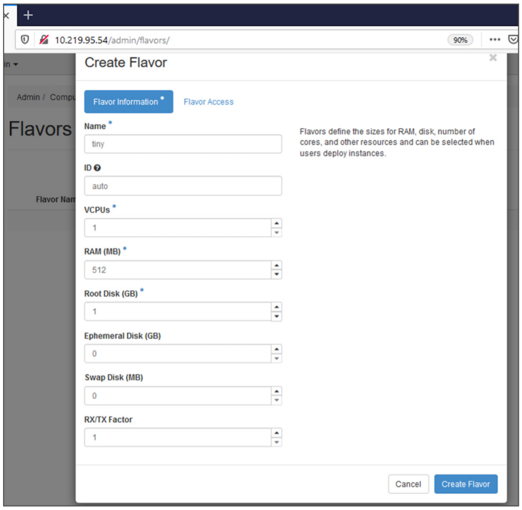


Figure 4.14 Populating Flavor Details

5. Once the information is filled in the Create Flavor form, click on the Create Flavor button.

Alternatively, you can also create a flavor through the CLI using this procedure:

```
openstack flavor --help
Command "flavor" matches:
  flavor create
  flavor delete
  flavor list
  flavor set
  flavor show
  flavor unset

[root@OPENSTACK kolla-toolbox]# openstack flavor create tiny --ram 512 --disk 1 --vcpus 1

+----------------------------+--------------------------------------+
| Field                      | Value                                |
+----------------------------+--------------------------------------+
| OS-FLV-DISABLED:disabled   | False                                |
| OS-FLV-EXT-DATA:ephemeral  | 0                                    |
| disk                       | 1                                    |
| id                         | f01091a5-24bd-4ddd-b6c2-78f6beebd627 |
| name                       | tiny                                 |
| os-flavor-access:is_public | True                                 |
| ram                        | 512                                  |
| rxtx_factor                | 1.0                                  |
| swap                       |                                      |
| vcpus                      | 1                                    |
+----------------------------+--------------------------------------+
```

Images

Image or *Virtual Machine image* is a single file which contains a virtual disk that has a bootable operating system installed on it. Image comes in different formats. For example, qcow2, iso, vmdk, etc.

A format describes the way the bits making up a file are arranged on the storage medium. Knowledge of a format is required in order for a consumer to interpret the content of the file correctly (rather than to simply view it as a bunch of bits).

For spawning a VM in OpenStack, one must have an image to start with. Let's add an image to glance repository.

Adding Images

An image can be added by the user, which becomes a project specific image, or by an administrator, which can be shared across projects if configured accordingly.

For now, we will create a shared image.

1. Download tinycore iso from its website to the local system.
2. Click on Images tab under Admin, Compute.
3. Click on + Create Image button.
4. Enter the image. For example: tinycore
5. Click on the Browse Button to select an image from local drive. Select the image that you downloaded.
6. Select the format as ISO – Optical Disk Image.
7. Leave other fields to default.
8. Click on Public for Visibility.
9. Click the Create Image button.

It will take few moments to upload the file to the OpenStack controller. Once uploaded the image will be similar to Figure 4.15.

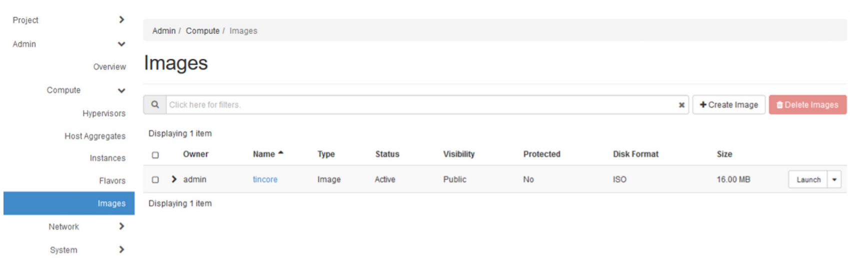


Figure 4.15 Uploading Images

Now that we've set up the basic environment of OpenStack for users to work on, let's log out from Horizon as administrator and log in back as a user of the Dayone tenant we created earlier.

Click on user Admin on the left top corner and click on Sign Out (Figure 4.16).

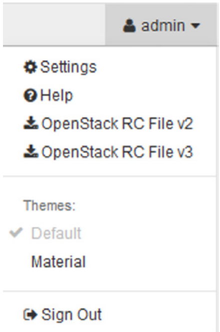


Figure 4.16 User Options Menu

Virtual Networks

The tenant's private network is called a *virtual network* (VN). A tenant of a project can have several VNs. Also, a VM can be connected to one or several VNs. VN's are the basic building blocks of the Contrail approach. Logical constructs implemented on top of the physical networks, VN's are used to replace VLAN-based isolation and provide multi-tenancy in a virtualized data center. Each tenant or an application can have one or more VNs. Each VN is isolated from all the other VNs unless explicitly allowed by security policy.

Virtual networks can also be implemented using two networks – a physical underlay network and a virtual overlay network.

The vRouters running in the hypervisors of the virtualized servers create a virtual overlay network on top of the physical underlay network using a mesh of dynamic "tunnels" amongst themselves.

The role of the physical underlay network is to provide an "IP fabric" – its responsibility is to provide unicast IP connectivity from any physical device (server, storage device, router, or switch) to any other physical device.

The vRouters, on the other hand, do contain per tenant state. They contain a separate forwarding table (a routing-instance) per VN (see Figure 4.17). That forwarding table contains the IP prefixes (in the case of Layer 3 overlays) or the MAC addresses (in the case of Layer 2 overlays) of the VMs. No single vRouter needs to contain all IP prefixes or all MAC addresses for all VMs in the entire data center. A given vRouter only needs to contain those routing instances that are locally present on the server (i.e., those have at least one VM present on the server.)

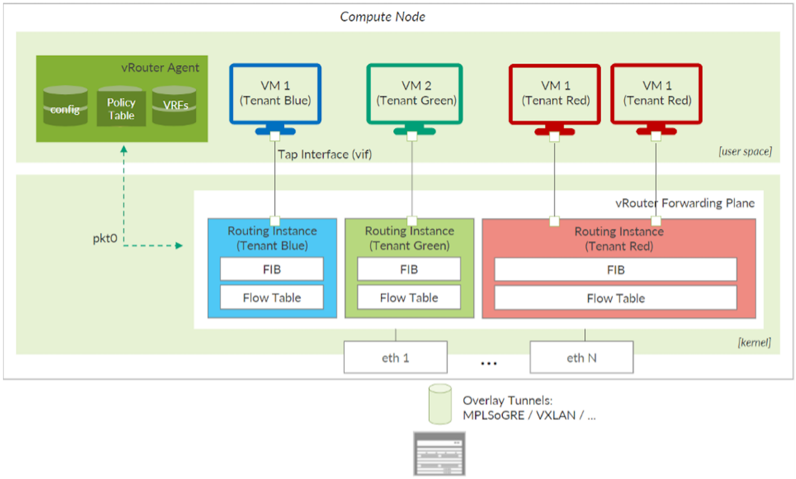


Figure 4.17 VN Implementation on vRouter Using Routing-instances and VRFs

Implementation of Virtual Networks

Contrail represents a VN using the format:

```
default-domain:<project ID>:<VNname>:<VRFname>
Example: default-domain:Dayone:IntranetVN:IntranetVN
```

Virtual networks in Contrail are implemented using a VRF on the vRouter for each VN hosted on that compute.

MORE? For more details on vRouter functioning, please refer to Tungsten Fabic architecture guide at https://tungstenfabric.github.io/website/Tungsten-Fabric-Architecture.html.

Create a VN Using CLI

1. Access the OpenStack kolla toolbox container.
2. Create a VN using the `openstack network create <VN NAME>` command:

```
(kolla-toolbox)[ansible@OPENSTACK /var/lib/kolla/config_files]$ openstack network create VNWeb
+---------------------------+--------------------------------------+
| Field                     | Value                                |
+---------------------------+--------------------------------------+
| admin_state_up            | UP                                   |
| availability_zone_hints   | None                                 |
| availability_zones        | None                                 |
| created_at                | None                                 |
| description               |                                      |
```

```
| dns_domain                | None                                 |
| id                        | d378bb75-792d-4d38-8a20-74084bb49893 |
| ipv4_address_scope        | None                                 |
| ipv6_address_scope        | None                                 |
| is_default                | None                                 |
| is_vlan_transparent       | None                                 |
| mtu                       | None                                 |
| name                      | VNWeb                                |
| port_security_enabled     | True                                 |
| project_id                | 0dab13dcc73f4e9ea553654998441e46     |
| provider:network_type     | None                                 |
| provider:physical_network | None                                 |
| provider:segmentation_id  | None                                 |
| qos_policy_id             | None                                 |
| revision_number           | None                                 |
| router:external           | Internal                             |
| segments                  | None                                 |
| shared                    | False                                |
| status                    | ACTIVE                               |
| subnets                   |                                      |
| tags                      |                                      |
| updated_at                | None                                 |
+---------------------------+--------------------------------------+
```

Create a Subnet and Attach to the VN Using CLI

1. Attach a subnet to the VN created earlier using the command:

`OpenStack subnet create <subnet name> --subnet-range <subnet id/prefix> --network <VN to attach>:`

```
(kolla-toolbox)[ansible@OPENSTACK /var/lib/kolla/config_
files]$ openstack subnet create WebSubnet --subnet-range 10.10.1.0/24 --network VNWeb
+-------------------------+--------------------------------------+
| Field                   | Value                                |
+-------------------------+--------------------------------------+
| allocation_pools        | 10.10.1.2-10.10.1.254                |
| cidr                    | 10.10.1.0/24                         |
| created_at              | None                                 |
| description             | None                                 |
| dns_nameservers         |                                      |
| enable_dhcp             | True                                 |
| gateway_ip              | 10.10.1.1                            |
| host_routes             |                                      |
| id                      | f254dd8b-1834-46ea-8590-04d86754c87b |
| ip_version              | 4                                    |
| ipv6_address_mode       | None                                 |
| ipv6_ra_mode            | None                                 |
| name                    | WebSubnet                            |
| network_id              | d378bb75-792d-4d38-8a20-74084bb49893 |
| project_id              | 0dab13dcc73f4e9ea553654998441e46     |
| revision_number         | None                                 |
| segment_id              | None                                 |
| service_types           | None                                 |
| subnetpool_id           | None                                 |
| tags                    |                                      |
| updated_at              | None                                 |
| use_default_subnet_pool | None                                 |
+-------------------------+--------------------------------------+
```

Gather Info about the VN Using the CLI

```
(kolla-toolbox)[ansible@OPENSTACK /var/lib/kolla/config_files]$ openstack network list
+--------------------------------------+--------------------------------+--------------------------------------+
| ID                                   | Name                           | Subnets                              |
+--------------------------------------+--------------------------------+--------------------------------------+
| 2c8d7a50-3fdd-4de5-b3c5-8d9ab4b217b0 | default-virtual-network        |                                      |
| a9fbff23-f237-4d86-88f4-d7bf31e991dc | ip-fabric                      |                                      |
| f8940f64-cd0e-4f4c-8e63-c7e61f97c690 | __link_local__                 |                                      |
| 8f0f3e20-30a8-4d4b-b795-dfcefb5cc4bd | _internal_vn_ipv6_link_local | e7c278dc-a186-11ea-a1de-525400d9bd15|
| d378bb75-792d-4d38-8a20-74084bb49893 | VNWeb                          | f254dd8b-1834-46ea-8590-04d86754c87b |
| 41c161ba-bfe6-4549-858b-7ec52831f31f | dci-network                    |                                      |
+--------------------------------------+--------------------------------+--------------------------------------+

(kolla-toolbox)[ansible@OPENSTACK /var/lib/kolla/config_files]$ openstack network show VNWeb
+---------------------------+--------------------------------------+
| Field                     | Value                                |
+---------------------------+--------------------------------------+
| admin_state_up            | UP                                   |
| availability_zone_hints   | None                                 |
| availability_zones        | None                                 |
| created_at                | None                                 |
| description               |                                      |
| dns_domain                | None                                 |
| id                        | d378bb75-792d-4d38-8a20-74084bb49893 |
| ipv4_address_scope        | None                                 |
| ipv6_address_scope        | None                                 |
| is_default                | None                                 |
| is_vlan_transparent       | None                                 |
| mtu                       | None                                 |
| name                      | VNWeb                                |
| port_security_enabled     | True                                 |
| project_id                | 0dab13dcc73f4e9ea553654998441e46     |
| provider:network_type     | None                                 |
| provider:physical_network | None                                 |
| provider:segmentation_id  | None                                 |
| qos_policy_id             | None                                 |
| revision_number           | None                                 |
| router:external           | Internal                             |
| segments                  | None                                 |
| shared                    | False                                |
| status                    | ACTIVE                               |
| subnets                   | f254dd8b-1834-46ea-8590-04d86754c87b |
| tags                      |                                      |
| updated_at                | None                                 |
+---------------------------+--------------------------------------+

(kolla-toolbox)[ansible@OPENSTACK /var/lib/kolla/config_files]$ openstack subnet show f254dd8b-1834-
46ea-8590-04d86754c87b
+-------------------------+--------------------------------------+
| Field                   | Value                                |
+-------------------------+--------------------------------------+
| allocation_pools        | 10.10.1.2-10.10.1.254                |
| cidr                    | 10.10.1.0/24                         |
| created_at              | None                                 |
| description             | None                                 |
| dns_nameservers         |                                      |
| enable_dhcp             | True                                 |
```

```
| gateway_ip             | 10.10.1.1                            |
| host_routes            |                                      |
| id                     | f254dd8b-1834-46ea-8590-04d86754c87b |
| ip_version             | 4                                    |
| ipv6_address_mode      | None                                 |
| ipv6_ra_mode           | None                                 |
| name                   | WebSubnet                            |
| network_id             | d378bb75-792d-4d38-8a20-74084bb49893 |
| project_id             | 0dab13dcc73f4e9ea553654998441e46     |
| revision_number        | None                                 |
| segment_id             | None                                 |
| service_types          | None                                 |
| subnetpool_id          | None                                 |
| tags                   |                                      |
| updated_at             | None                                 |
| use_default_subnet_pool | None                               |
+------------------------+--------------------------------------+
```

Create Virtual Networks Using the OpenStack GUI

1. Log back in as User1 to Horizon as in Figure 4.18.

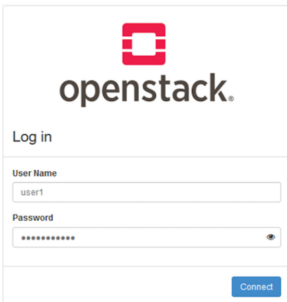


Figure 4.18 OpenStack Login Screen

Note that since Dayone is the only project that is enabled for this user, the project drop list will only have one project listed as shown in Figure 4.19.

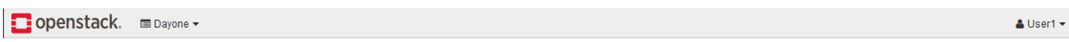


Figure 4.19 OpenStack Project Navigation and User Actions Bar

Expand the Project tab, click on network, and click on the Networks subtab. Click Create Network and in the Create Network form, enter the VNWeb. Leave the check boxes to their default state. Click Next.

Now enter the Subnet Name (example: Web1) and then enter the network address (example: 10.1.0.0/24). Click on next. Then click on Create.

Creating Virtual Networks Through Contrail GUI

Note that in the Contrail GUI interface, usually virtual networks are simply called *networks*.

1. Log in to Contrail GUI.
2. Click configure tab icon, select Networking subtab, select Networks.
3. Make sure the Dayone project is selected since the VN that we would like to create should belong to this tenant.
4. Click on + .
5. In the Create form, enter the VN name as *VNApp*.
6. Expand the subnets section by click on the arrow.
7. Click on + to add a new subnet to VNApp.
8. Select default-network-ipam under the IPAM list.
9. Enter 10.10.2.0/24 in CIDR section.
10. Leave Allocation pools to default. You can enter a range if you would like to assign IP address to instances only in a range.
11. Make sure the gateway checkbox is selected and the gateway IP address is the first IP address of the subnet.
12. Leave the service address to default state. This address will be used by vRouter to service DNS requests.
13. Make sure that the DNS and DHCP check boxes are selected.
14. Click Save.

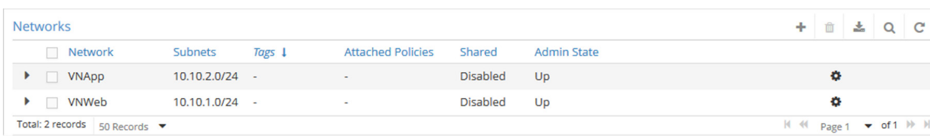


Figure 4.20 Virtual Networks in Contrail GUI

Repeat to create these following networks:

VN Name	Subnet/CIDR	Subnet name (only in Openstack GUI)
IntranetVN	10.10.100.0/24	Intranet1
VMMgmt	10.10.0.0/24	MGMT1
InternetVN	**203.0.113.32/28**	**Internet1**
DBVN	10.10.3.0/24	DB1

And the creation process looks like the topology in Figure 4.21.

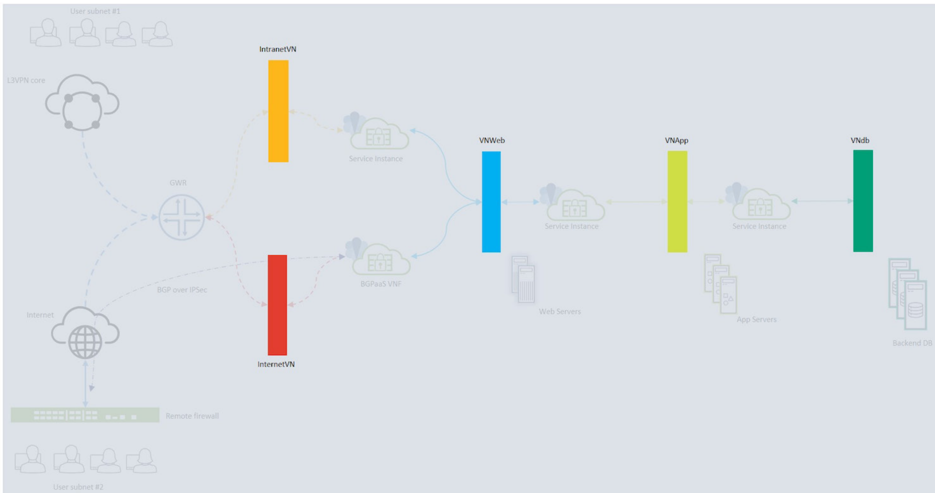


Figure 4.21 Three-tier Application Creation Progress After Creating VNs

Now that the VNs are created, let's spawn some VMs that will be connected to these networks to communicate between them. Note that by default, VMs in one VN cannot communicate with other VNs.

Launch a VM Through OpenStack GUI

1. Select the Dayone project.
2. Click on Project tab.
3. Click on Compute.
4. Click on Instances.
5. Click Launch Instance button.

6. Enter instance name in Details tabs as WebServer1 (see Figure 4.22).

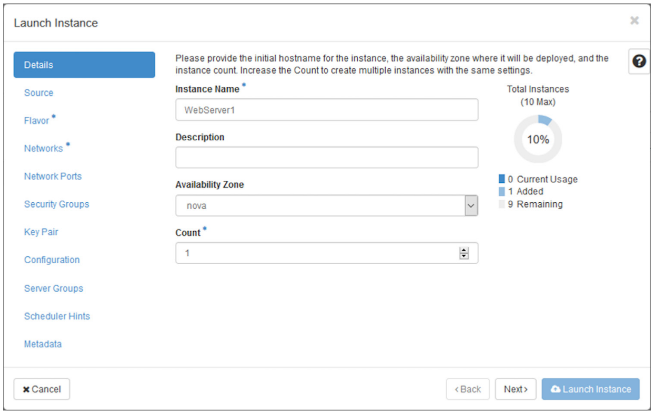


Figure 4.22 VM Creation Through OpenStack GUI

7. Click on Source tab.
8. Click the Up Arrow icon in the tinycore image row to select image. Once clicked the tinycore image should appear in the Allocated section of the screen.
9. Click Flavor tab.
10. Click the Up Arrow icon in the tiny flavor we created in an earlier section.
11. Click the Networks tab.
12. Click the Up Arrow icon for the *VNWeb*.
13. Click the Launch Instance button.

It will take few moments to instantiate the VM. Once completed the status will show *Running* (see Figure 4.23). Take a moment to observe the instance name, image name, IP address, Flavor, and power state.

The first IP address of the subnet is used by the default gateway and the second IP address is used for providing services such as DNS.

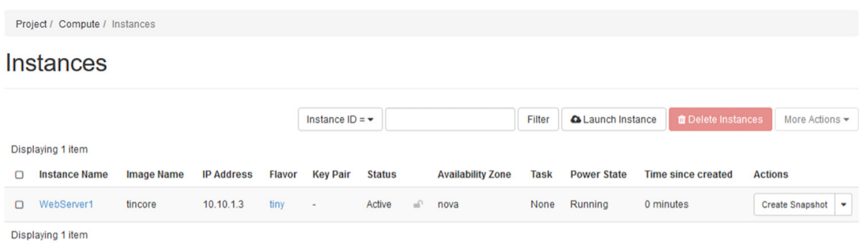


Figure 4.23 Project > Compute > Instances Screen

Repeat to create VMs as per the following table.

VM Name	Count	Image	Flavor	Associated network
WebServer2	1	Cirros	tiny	VNWeb
AppServer	3	Cirros	tiny	VNApp
DBServer	3	Cirros	tiny	DbVN
Test1	1	Cirros	Tiny	IntranetVN

And the creation process will look like the topology in Figure 4.24.

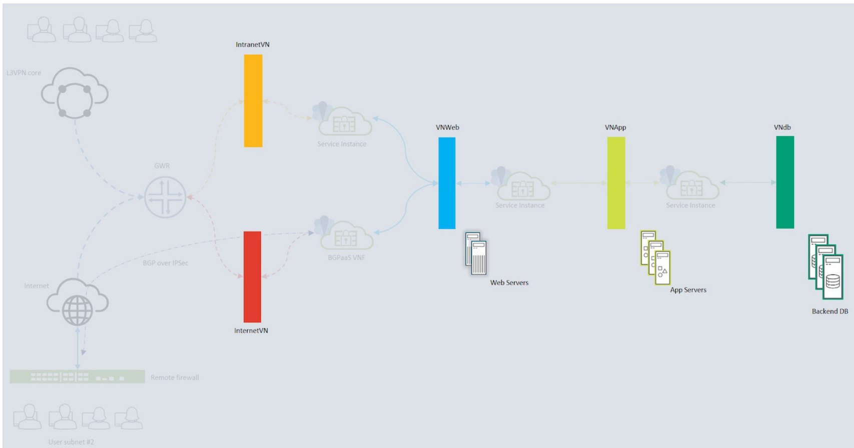


Figure 4.24 Three-tier Application Creation Progress After Spawning/Launching VMs

Life of a Packet Within a Virtual Network

In this section, let's use the Contrail vRouter CLI to trace the path between WebServer1 and WebServer2.

To access the routing table on a containerized contrail compute, run the `docker ps` command and identify the ID of container named contrail-vroute-agent:xxxxx.

1. Access the shell of the container by running the `docker exec -it <contrail id> /bin/bash` command.

NOTE Sometimes the prompt may wrap the commands you are typing. To prevent this, you can add `-e COLUMNS=$COLUMNS -e LINES=$LINES` to export the host column and line variable to the container prompt:

```
[root@compute2 ~]# docker ps
CONTAINER ID       IMAGE                                                      COMMAND              CREATED
STATUS             PORTS              NAMES
711f9aac1834       hub.juniper.net/contrail/contrail-vrouter-agent:2003.1.40      "/entrypoint.sh /usrâ€¦"
2 months ago       Up 13 days                         vrouter_vrouter-agent_1
59c5f60fcf90       hub.juniper.net/contrail/contrail-nodemgr:2003.1.40          "/entrypoint.sh /binâ€¦"
2 months ago       Up 13 days                         vrouter_nodemgr_1
07a7f5a51d44       hub.juniper.net/contrail/contrail-provisioner:2003.1.40       "/entrypoint.sh /usrâ€¦"
2 months ago       Up 13 days                         vrouter_provisioner_1
b3a9db1c428e       hub.juniper.net/contrail/contrail-external-rsyslogd:2003.1.40  "/contrail-entrypoinâ€¦"
2 months ago       Up 13 days                         rsyslogd_rsyslogd_1
26343be2cb16       kolla/centos-binary-nova-compute:queens                    "dumb-init --single-â€¦"   2
months ago       Up 13 days                         nova_compute
5cf2f32e8ce8       kolla/centos-binary-nova-libvirt:queens                    "dumb-init --single-â€¦"   2
months ago       Up 13 days                         nova_libvirt
848f7f67a71a       kolla/centos-binary-nova-ssh:queens                        "dumb-init --single-â€¦"   2
months ago       Up 13 days                         nova_ssh
ccbbe0bd9ddf       kolla/centos-binary-cron:queens                            "dumb-init --single-â€¦"   2
months ago       Up 13 days                         cron
8c383a2d59a6       kolla/centos-binary-kolla-toolbox:queens                   "dumb-init --single-â€¦"   2
months ago       Up 13 days                         kolla_toolbox
f23c9dfacbb9       kolla/centos-binary-fluentd:queens                         "dumb-init --single-â€¦"   2
months ago       Up 13 days                         fluentd
[root@compute2 ~]#

[root@compute2 ~]# docker exec -ti -e COLUMNS=$COLUMNS -e LINES=$LINES 711f9aac1834 /bin/bash
(vrouter-agent)[root@compute2 /]$
```

The Life of Packet in a Virtual Network Across Compute Nodes

Let us now explore the life of packet when ping is initiated from a VM on compute 1 to a VM on compute2:

1. Initiate ping from WebServer1 and WebServer2 and let run while we are exploring the various outputs to understand the life of packet.
2. Access the agent container shell using docker exec -it <docker id> bash.
3. Run the command vif --list.
4. Observe the output of vif0/3 which is connected to source VM:

```
vif0/3      OS: tape4bf1dc8-09 NH: 29
            Type:Virtual HWaddr:00:00:5e:00:01:00 IPaddr:10.10.1.3
            Vrf:2 Mcast Vrf:2 Flags:PL3L2DEr QOS:-1 Ref:6
            RX packets:777178  bytes:32642076 errors:0
            TX packets:777704  bytes:32664170 errors:0
            ISID: 0 Bmac: 02:e4:bf:1d:c8:09
            Drops:0
```

A few notes on this output:

- Here the vif0/3 is connected to a TAP interface tape4bf1dc8-09 on the other side, which is in turn connected a VM's vNIC. This interface is also L3, L2 capable and is policy enabled. When an interface is policy enabled, traffic to and from it will require the vRouter to policy/flow lookup for forwarding the traffic. This can be disabled through configuration if required.

- D in the flags section suggests that the interface will be provided with DHCP IP address upon request from vNIC.
- This vif interface is part of VRF 2.
- Also notice the NH for this interface is 29.

5. Now let's use the flow list command to get the list of flows from compute1:`(vrouter-agent)[root@compute1 /]$` **`flow -l`**

```
Flow table(size 161218560, entries 629760)

Entries: Created 112 Added 112 Deleted 116 Changed 132Processed 112 Used Overflow entries 0
(Created Flows/CPU: 1 0 1 1 2 1 2 5 1 1 2 0 7 2 10 0 5 3 4 1 2 1 1 1 2 1 0 7 5 0 3 0 1 0 1 0 0 0 1 18 1 8 0 0
0 0 0 0 2 0 7 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0)(oflows 0)

Action:F=Forward, D=Drop N=NAT(S=SNAT, D=DNAT, Ps=SPAT, Pd=DPAT, L=Link Local Port)
 Other:K(nh)=Key_Nexthop, S(nh)=RPF_Nexthop
 Flags:E=Evicted, Ec=Evict Candidate, N=New Flow, M=Modified Dm=Delete Marked
TCP(r=reverse):S=SYN, F=FIN, R=RST, C=HalfClose, E=Established, D=Dead

    Index                Source:Port/Destination:Port                      Proto(V)
-----------------------------------------------------------------------------------
    31508<=>162036       10.10.1.5:63299                                     6 (2)
                         10.10.1.1:179
(Gen: 1, K(nh):83, Action:F, Flags:, TCP:S, QOS:-1, S(nh):83,  Stats:9/606,
 SPort 60282, TTL 0, Sinfo 9.0.0.0)

--snip--

   317536<=>385592       10.10.1.3:9986                                      1 (2)
                         10.10.1.4:0
(Gen: 1, K(nh):29, Action:F, Flags:, QOS:-1, S(nh):29,  Stats:5/490,  SPort 62691,
 TTL 0, Sinfo 3.0.0.0)

--snip--

   385592<=>317536       10.10.1.4:9986                                      1 (2)
                         10.10.1.3:0
(Gen: 1, K(nh):29, Action:F, Flags:, QOS:-1, S(nh):34,  Stats:5/490,  SPort 62567,
 TTL 0, Sinfo 192.168.110.20)

--snip--

(vrouter-agent)[root@compute1 /]$
```

You can see from the output that it can determine that the flows are successfully created, and packets are being forwarded based on the action flag which is set to F.

Let's check the path the packet is taking to reaching compute node2 and the destination VM. To do this, get the route table output for 10.10.1.4/32, which is the destination VM spun on compute 2:

```
(vrouter-agent)[root@compute1 /]$ rt --dump 2 | egrep "Dest|10.10.1.4/32"
Destination          PPL        Flags        Label         Nexthop    Stitched MAC(Index)
10.10.1.4/32          32           LP         51             34        2:5e:1e:ce:aa:98(202044)
```

The next hop for this destination is 34 and the label that will be used while encapsulating the packet is 51.

Now let's check the next hop to understand the path packet is going to take:

```
(vrouter-agent)[root@compute1 /]$ nh --get 34
Id:34          Type:Tunnel         Fmly: AF_INET  Rid:0  Ref_cnt:15         Vrf:0
               Flags:Valid, MPLSoUDP, Etree Root,
               Oif:0 Len:14 Data:f8 f2 1e 79 66 90 f8 f2 1e 79 5b d0 08 00
               Sip:192.168.110.19 Dip:192.168.110.20
```

This output indicates the packet is going to tunnel output to 192.168.110.20 using MPLSoUDP encapsulation, as well as the following information:

- Type: Tunnel
- Oif: 0
- SIP: 192.168.110.19
- DIP: 192.168.110.20
- Flags: MPLSoUDP

NOTE In some cases, Nova can schedule the VMs to spin on the same compute. In such cases the NH will point to a local interface.

Let's explore the outputs on compute2:

```
(vrouter-agent)[root@compute2 /]$ mpls --get 51
MPLS Input Label Map

   Label    NextHop
-------------------
      51       64
(vrouter-agent)[root@compute2 /]$ nh --get 64
Id:64          Type:Encap          Fmly: AF_INET  Rid:0  Ref_cnt:4          Vrf:5
               Flags:Valid, Policy, Etree Root,
               EncapFmly:0806 Oif:7 Len:14
               Encap Data: 02 5e 1e ce aa 98 00 00 5e 00 01 00 08 00

(vrouter-agent)[root@compute2 /]$ vif --get 7
Vrouter Interface Table

Flags: P=Policy, X=Cross Connect, S=Service Chain, Mr=Receive Mirror
      Mt=Transmit Mirror, Tc=Transmit Checksum Offload, L3=Layer 3, L2=Layer 2
      D=DHCP, Vp=Vhost Physical, Pr=Promiscuous, Vnt=Native Vlan Tagged
      Mnp=No MAC Proxy, Dpdk=DPDK PMD Interface, Rfl=Receive Filtering Offload, Mon=Interface is
Monitored
      Uuf=Unknown Unicast Flood, Vof=VLAN insert/strip offload, Df=Drop New Flows, L=MAC Learning
```

```
Enabled
     Proxy=MAC Requests Proxied Always, Er=Etree Root, Mn=Mirror without Vlan Tag, HbsL=HBS Left Intf
     HbsR=HBS Right Intf, Ig=Igmp Trap Enabled

vif0/7      OS: tap5e1eceaa-98 NH: 64
            Type:Virtual HWaddr:00:00:5e:00:01:00 IPaddr:10.10.1.4
            Vrf:5 Mcast Vrf:5 Flags:PL3L2DEr QOS:-1 Ref:6
            RX packets:777879  bytes:32712342 errors:0
            TX packets:778329  bytes:32731244 errors:0
            Drops:0
```

Notice that the vrf ID for VNWeb differs between compute1 and compute2. In Contrail, it does not matter if the vrf for the same VN across computes differ, or if the vrf ID of different computes are matching on two different computes. What matters is how labels are mapped to vrf on the receiving side.

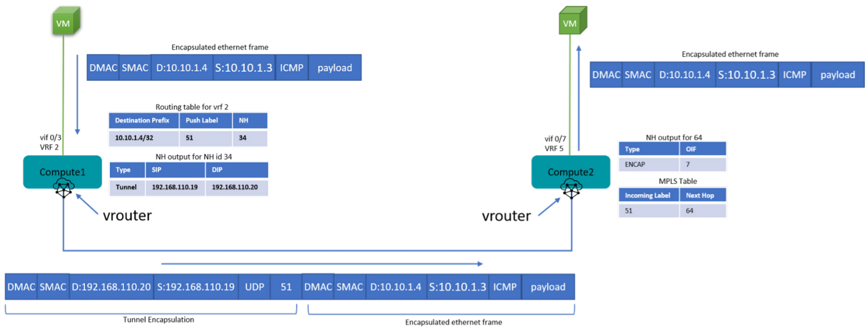


Figure 4.25 *Life of Packet Between VMs of the Same VN on Different Computes*

Accessing VNs Across WAN

Once we have services hosted in the DC it is desirable to provide access to these services from outside the cloud. There are several methods for doing this, such as SNAT, floating IP addresses, or by advertising the VN to a DC gateway router that the VNs can reach from external subnets. This router can provide connectivity to the internet or to private networks through technologies like L3VPN, dedicated circuit, etc.

Contrail needs to be configured to peer with one or more of the routers designated as DC gateway routers. If there are three controllers, then the DCGW will need to be configured with all three controller IPs as peers. Routes matching the RT between VNs configured in Contrail and VRFs on DCGW are exchanged. This is exactly the same as the two PE device exchange routes in L3VPN.

The vRouter and the DCGW will form MPLSoGRE dynamic tunnels (see Figure 4.26) using the routes and next hops learned from the controller.

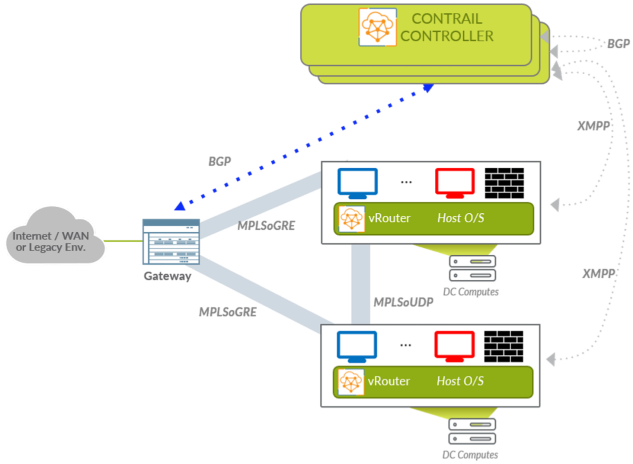


Figure 4.26 BGP Peering Between DCGW and Controller

Configuration Required on Contrail GUI

To add a BGP router follow these steps that refer to Figures 4.27a and 4.27b.

1. Access the Contrail GUI.
2. Click on the Settings icon to configure.
3. Click on Expand Infrastructure.
4. Click on BGP Routers. This will open a window to configure and BGP Physical routers for this contrail cluster.
5. Note: the list will be already populated with the controllers as BGP routers.
6. Click on the + icon for a new BGP router.
7. Enter details such as hostname, vendor ID, IP address for peering, router ID, AS number, and BGP router ASN.

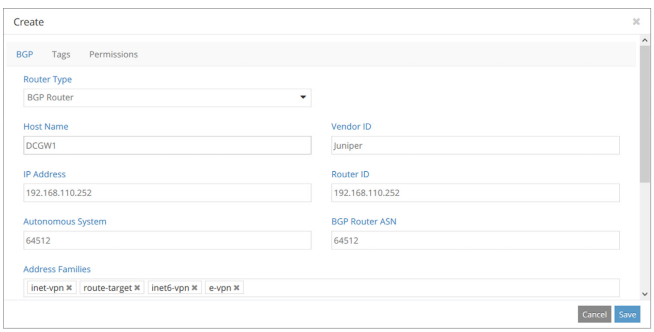


Figure 4.27a BGP Router Create Screen #1

7. Scroll down on the create screen to expand the associated peer's section.

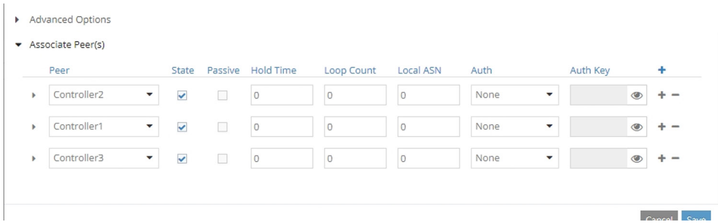


Figure 4.27b BGP Router Create Screen #2

8. Click on the + icon three times in the peer section.
9. Select a controller in each of the peer drop down list and leave the other values at their default.
10. Click save.

Configuring Intranet VN with RT Matching the DCGW Intranet VRF Import RT

1. Navigate to Configure\Networks.
2. In the networks screen, click on the gear icon of the IntranetVN.
3. Click on the edit option which appears after click on gear icon.
4. Scroll down to Route Target(s) option.
5. Click on the + icon.
6. Enter the ASN and Target matching the import RT value of the DCGW. See Figure 4.28. In our example, ASN is 64512 and Target is 10000.

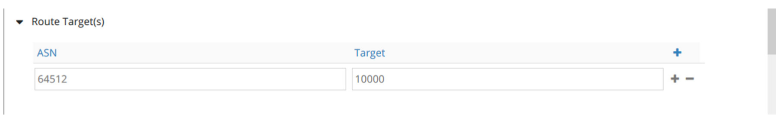


Figure 4.28 Configuring Intranet VN with RT Matching

7. Click Save.

Configuration on the Gateway Router

```
set interfaces lt-0/0/0 unit 0 encapsulation frame-relay
set interfaces lt-0/0/0 unit 0 dlci 1
set interfaces lt-0/0/0 unit 0 peer-unit 1
set interfaces lt-0/0/0 unit 0 family inet

set interfaces lt-0/0/0 unit 1 encapsulation frame-relay
set interfaces lt-0/0/0 unit 1 dlci 1
set interfaces lt-0/0/0 unit 1 peer-unit 0
set interfaces lt-0/0/0 unit 1 family inet

set interfaces xe-0/0/0 unit 0 family inet address 192.168.10.252/24

set interfaces lo0 unit 1 family inet address 192.0.2.1/24
set routing-instances Intranet interface lo0.1

set routing-options static route 10.10.1.0/24 next-hop lt-0/0/0.0
set routing-options route-distinguisher-id 192.168.10.252
set routing-options autonomous-system 64512
set routing-options dynamic-tunnels dynamic_overlay_tunnels source-address 192.168.10.252
set routing-options dynamic-tunnels dynamic_overlay_tunnels gre
set routing-options dynamic-tunnels dynamic_overlay_tunnels destination-networks 192.168.10.0/24

set protocols mpls interface all

set protocols bgp group contrail type internal
set protocols bgp group contrail local-address 192.168.10.252
set protocols bgp group contrail keep all
set protocols bgp group contrail family inet-vpn unicast
set protocols bgp group contrail neighbor 192.168.10.55
set protocols bgp group contrail neighbor 192.168.10.56
set protocols bgp group contrail neighbor 192.168.10.57

set routing-instances Intranet instance-type vrf
set routing-instances Intranet interface lt-0/0/0.1
set routing-instances Intranet vrf-target target:64512:10000
set routing-instances Intranet routing-options static route 0.0.0.0/0 next-hop lt-0/0/0.1
```

Verifying Tables on the Gateway Router and Contrail Controller

```
root@DCGW1> show bgp summary
Groups: 1 Peers: 3 Down peers: 0
Table          Tot Paths  Act Paths Suppressed    History Damp State    Pending
inet.0
                      0          0          0          0          0          0
bgp.l3vpn.0
                     36         18          0          0          0          0
Peer                AS      InPkt     OutPkt    OutQ   Flaps Last Up/Dwn State|#Active/Received/
Accepted/Damped...
192.168.110.55       64512         20          3       0       2           7 Establ
  bgp.l3vpn.0: 18/18/18/0
192.168.110.56       64512         20          4       0       2           7 Establ
  bgp.l3vpn.0: 0/18/18/0
192.168.110.57       64512          2          3       0       2           7 Establ
  bgp.l3vpn.0: 0/0/0/0
root@DCGW1>
```

Figure 4.29 Logical View of BGP Peering Between Contrail Controller and DC Gateway Router

Life of Packet from GWR to VM Belonging to Intranet VN

1. Access the DCGW using SSH and initiate a ping to 10.10.100.3:

```
root@DCGW1> ping 10.10.100.3 routing-instance Intranet source 192.0.2.1 rapid count 5
PING 10.10.100.3 (10.10.100.3): 56 data bytes
.....
--- 10.10.100.3 ping statistics ---
5 packets transmitted, 0 packets received, 100% packet loss
```

If you observe this output, the ping was not successful, so let's trace the packet and understand the cause of the failure.

2. Create another session with DCGW.

3. Check the routing table Intranet.inet.0 on DCGW:

```
root@DCGW1> show route table Intranet.inet.0

Intranet.inet.0: 4 destinations, 5 routes (4 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

0.0.0.0/0          *[Static/5] 14w5d 13:32:57
                    > via lt-0/0/0.1
10.10.100.3/32     *[BGP/170] 9w6d 14:55:54, MED 100, localpref 200, from 192.168.110.56
                      AS path: ?, validation-state: unverified
                    > via gr-0/0/0.32769, Push 25
                    [BGP/170] 11w6d 00:03:07, MED 100, localpref 200, from 192.168.110.57
                      AS path: ?, validation-state: unverified
                    > via gr-0/0/0.32769, Push 25
192.0.2.0/24       *[Direct/0] 14w4d 16:03:21
                    > via lo0.1
192.0.2.1/32       *[Local/0] 14w4d 16:03:21
                      Local via lo0.1
```

4. Observe here that the destination IP address is reachable through gr-0/0/0.32769 on the active path with push label as 25. The “from” IP address here is of the controller which is also working as route reflector.

5. Collect the outputs for show route inet.3. Here we can notice that the gr-0/0/0.32769 is pointing towards 192.168.110.20/32 which is the fabric IP address of compute 2:

```
inet.3: 3 destinations, 3 routes (3 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

192.168.110.0/24   *[Tunnel/300] 21w0d 22:09:25
                      Tunnel
192.168.110.19/32  *[Tunnel/300] 00:09:14
                    > via gr-0/0/0.32771
192.168.110.20/32  *[Tunnel/300] 14w3d 05:10:35
                    > via gr-0/0/0.32769
```

6. Verify the dynamic-tunnel database for identifying source and destination IP addresses of GRE tunnel towards compute2:

```
root@DCGW1> show dynamic-tunnels database
Table: inet.3

Destination-network: 192.168.110.0/24
Tunnel to: 192.168.110.19/32 State: Up (expires in 00:07:27 seconds)
  Reference count: 0
  Next-hop type: gre
    Source address: 192.168.110.252
    Next hop: gr-0/0/0.32771
    State: Up
Tunnel to: 192.168.110.20/32 State: Up
  Reference count: 1
  Next-hop type: gre
    Source address: 192.168.110.252
    Next hop: gr-0/0/0.32769
    State: Up
```

7. In step 3, we noted that the DCGW is going to send the encapsulated packet with the label 25. To check the mapping of incoming label 25 to its NH on compute 2, run the command `mpls --get 25` on the agent container shell:

```
(vrouter-agent)[root@compute2 /]$ mpls --get 25
MPLS Input Label Map

   Label    NextHop
-------------------
      25       30
```

Label 25 is mapped to NH 30.

8. Let's check the NH 30 outgoing interface(oif):

```
(vrouter-agent)[root@compute2 /]$ nh --get 30
Id:30         Type:Encap          Fmly: AF_INET  Rid:0  Ref_cnt:4         Vrf:2
              Flags:Valid, Policy, Etree Root,
              EncapFmly:0806 Oif:3 Len:14
              Encap Data: 02 f6 fd 73 62 6d 00 00 5e 00 01 00 08 00
```

9. And now check the vif 3 interface:

```
(vrouter-agent)[root@compute2 /]$ vif --get 3
Vrouter Interface Table

Flags: P=Policy, X=Cross Connect, S=Service Chain, Mr=Receive Mirror
       Mt=Transmit Mirror, Tc=Transmit Checksum Offload, L3=Layer 3, L2=Layer 2
       D=DHCP, Vp=Vhost Physical, Pr=Promiscuous, Vnt=Native Vlan Tagged
      Mnp=No MAC Proxy, Dpdk=DPDK PMD Interface, Rfl=Receive Filtering Offload, Mon=Interface is
Monitored
      Uuf=Unknown Unicast Flood, Vof=VLAN insert/strip offload, Df=Drop New Flows, L=MAC Learning
Enabled
      Proxy=MAC Requests Proxied Always, Er=Etree Root, Mn=Mirror without Vlan Tag, HbsL=HBS Left Intf
      HbsR=HBS Right Intf, Ig=Igmp Trap Enabled

vif0/3      OS: tapf6fd7362-6d NH: 30
            Type:Virtual HWaddr:00:00:5e:00:01:00 IPaddr:10.10.100.3
            Vrf:2 Mcast Vrf:2 Flags:PL3L2DEr QOS:-1 Ref:6
            RX packets:777721  bytes:32670930 errors:0
            TX packets:778417  bytes:32700148 errors:0
            ISID: 0 Bmac: 02:f6:fd:73:62:6d
            Drops:0
```

Based on the output verified so far, we can trace the path from DCGW to compute for a packet with source IP address 192.0.2.1 and destination IP address 10.10.100.3, as shown in Figure 4.30.

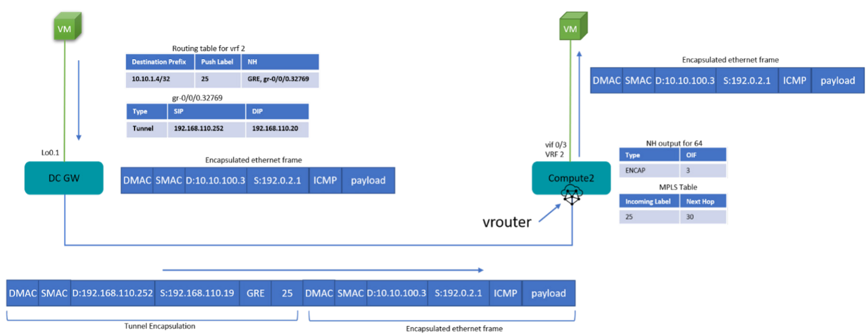


Figure 4.30 Life of Packet from DC GW Router to VM on Compute Node

So far everything seems to be as expected. Then why are packets are getting dropped? To find the answer let's use CLI tools such as `dropstats` and `flow` provided by the vRouter and fix the issue.

First let's see if there are any drops recorded by the vRouter:

```
(vrouter-agent)[root@compute2 /]$ dropstats
Invalid IF                  0
Trap No IF                  0
IF TX Discard               0
IF Drop                     0
IF RX Discard               0

Flow Unusable               0
Flow No Memory              0
Flow Table Full             0
Flow NAT no rflow           0
Flow Action Drop            14   <<<<
Flow Action Invalid         0
Flow Invalid Protocol       0
Flow Queue Limit Exceeded   0
New Flow Drops              0
Flow Unusable (Eviction)    0

Original Packet Trapped     0
```

Yes, indeed there are drops and they are categorized as `Flow Action`. This means vRouter is configured not to allow packets for this destination or from this source. This happens either due to security groups configuration or security policies. To dig deeper, you can look at the flow output and pinpoint the exact configuration that needs tuning:

```
(vrouter-agent)[root@compute2 /]$ flow -l
Flow table(size 161218560, entries 629760)

Entries: Created 5903 Added 5896 Deleted 11772 Changed 11821Processed 5901 Used Overflow entries 0
(Created Flows/CPU: 253 29 88 68 39 1 3 41 154 11 256 6 291 12 137 4 2764 2 172 373 161 50 94 0 40 148 61
11 27 5 30 0 47 45 60 0 0 0 396 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 5 0 3 0 7 0 0 0 0 0 6 0 1 0 0 0)(oflows 0)

Action:F=Forward, D=Drop N=NAT(S=SNAT, D=DNAT, Ps=SPAT, Pd=DPAT, L=Link Local Port)
 Other:K(nh)=Key_Nexthop, S(nh)=RPF_Nexthop
 Flags:E=Evicted, Ec=Evict Candidate, N=New Flow, M=Modified Dm=Delete Marked
TCP(r=reverse):S=SYN, F=FIN, R=RST, C=HalfClose, E=Established, D=Dead

    Index                Source:Port/Destination:Port                      Proto(V)
-----------------------------------------------------------------------------
   104580<=>158512       10.10.100.3:11428                                   1 (2)
                         192.0.2.1:0
(Gen: 1, K(nh):30, Action:H, Flags:, QOS:-1, S(nh):30,  Stats:0/0,  SPort 51000,
 TTL 0, Sinfo 0.0.0.0)

   155364<=>160332       192.0.2.1:11423                                     1 (2)
                         10.10.100.3:0
(Gen: 1, K(nh):30, Action:D(SG), Flags:, QOS:-1, S(nh):79,  Stats:53/4452,
 SPort 53468, TTL 0, Sinfo 192.168.110.252)

   158512<=>104580       192.0.2.1:11428                                     1 (2)
                         10.10.100.3:0
(Gen: 1, K(nh):30, Action:D(SG), Flags:, QOS:-1, S(nh):79,  Stats:5/420,
 SPort 49279, TTL 0, Sinfo 192.168.110.252)
```

Based on the flow outputs, we can determine that the flow action drop is being set due to SG or security groups.

Security Groups

Security groups are similar to stateless firewall filters in Junos. Using security groups, you can filter traffic based on the constraints mentioned in the rules. Security groups are applied on the vif interfaces, which come into effect after policy checks by the vRouter. Security groups rules have the following parameters to filter traffic:

- Direction(Ingress, Egress)
- EtherType (IPv4, IPv6)
- CIDR address
- Protocol
- Port Range

Let's modify the existing SG to allow traffic from other sources (Figure 4.31):

1. Navigate to configure / Networking / Security Groups in contrail UI.
2. Make sure the project selected in the project drop down list is Dayone.
3. Click on gear icon next to the default SG to edit it.
4. Edit the ingress and egress rules for IPv4 to reflect the address as 0.0.0.0/0 instead of "default".
5. Also make sure that the protocol is "ANY" for both ingress and egress rules. It's necessary to have the protocol as ANY so it can be configured per the security requirement of the tenant.

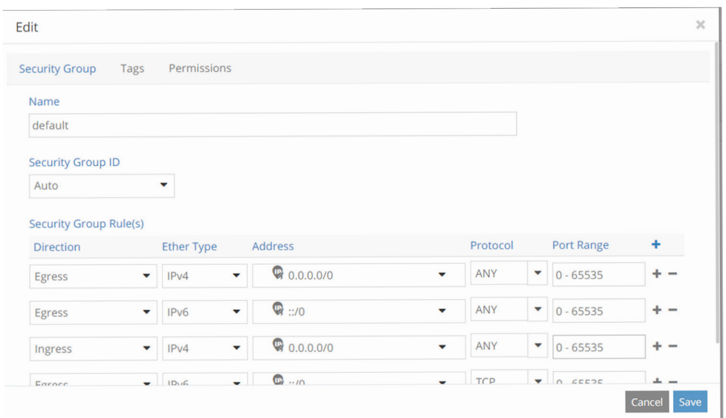


Figure 4.31 Security Groups Settings

6. Click on Save.

7. Switch back to the CLI of DCGW and initiate a ping to 10.10.100.3. You should be able to see ping working successfully.

```
root@DCGW1> ping 10.10.100.3 routing-instance Intranet source 192.0.2.1
PING 10.10.100.3 (10.10.100.3): 56 data bytes
64 bytes from 10.10.100.3: icmp_seq=0 ttl=63 time=1.462 ms
64 bytes from 10.10.100.3: icmp_seq=1 ttl=63 time=3.085 ms
```

8. Verify if the flow action drop counter has stopped incrementing. To avoid confusion, first clear all the dropstats counter and then execute to print the latest values:

```
(vrouter-agent)[root@compute2 /]$ dropstats --clear
Dropstats counters cleared successfully on all cores
(vrouter-agent)[root@compute2 /]$ dropstats
Invalid IF                    0
Trap No IF                    0
IF TX Discard                 0
IF Drop                       0
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
New Flow Drops                0
Flow Unusable (Eviction)      0
```

9. Also check the flow table to verify if the flow action is set to forward instead of "drop" as observed before enabling security groups.

In live environments, there could hundreds of active flows on vRouter. Instead of listing all these flows, you can simply request vRouter to match particular criteria:

```
e.g.: --match 1.1.1.1:20
      --match "1.1.1.1:20,2.2.2.2:22"
      --match "[fe80::225:90ff:fec3:afa]:22"
      --match "10.204.217.10:56910 & vrf 0 & proto tcp"
      --match "10.204.217.10:56910,169.254.0.3:22 & vrf 0 & proto tcp"
              proto {tcp, udp, icmp, icmp6, sctp}


(vrouter-agent)[root@compute2 /]$ flow --match 10.10.100.3
Flow table(size 161218560, entries 629760)

Entries: Created 5912 Added 5902 Deleted 11786 Changed 11834Processed 5912 Used Overflow entries 0
(Created Flows/CPU: 253 29 89 68 39 1 3 41 154 11 256 6 291 12 137 4 2770 2 172 373 161 50 94 0 40 148 61
11 27 5 30 0 47 45 60 0 2 0 396 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 5 0 3 0 7 0 0 0 0 0 6 0 1 0 0 0)(oflows 0)
```

```
Action:F=Forward, D=Drop N=NAT(S=SNAT, D=DNAT, Ps=SPAT, Pd=DPAT, L=Link Local Port)
 Other:K(nh)=Key_Nexthop, S(nh)=RPF_Nexthop
 Flags:E=Evicted, Ec=Evict Candidate, N=New Flow, M=Modified Dm=Delete Marked
TCP(r=reverse):S=SYN, F=FIN, R=RST, C=HalfClose, E=Established, D=Dead

Listing flows matching ([10.10.100.3]:*)

    Index                Source:Port/Destination:Port                      Proto(V)
-----------------------------------------------------------------------------------
     2596<=>475320       192.0.2.1:11539                                     1 (2)
                         10.10.100.3:0
(Gen: 1, K(nh):30, Action:F, Flags:, QOS:-1, S(nh):79,  Stats:19/1596,
 SPort 53391, TTL 0, Sinfo 192.168.110.252)

   475320<=>2596         10.10.100.3:11539                                   1 (2)
                         192.0.2.1:0
(Gen: 1, K(nh):30, Action:F, Flags:, QOS:-1, S(nh):30,  Stats:19/1862,
 SPort 60257, TTL 0, Sinfo 3.0.0.0)
```

Network Policies

Network policies are used to control the communication between virtual networks. In Contrail, users can define a simple policy by creating the policy and then create rules within a policy that will determine which protocol, networks, and port should be allowed or denied. The sequence of the rule within a policy dictates if the rule will be used to filter the traffic or not.

In this section we will create a simple policy that allows communication between WEB_VN and APP_VN using only the ICMP protocol.

Create a Policy

1. In the configuration menu, expand the networking section and click on policies. By default, there would not be any policies defined.

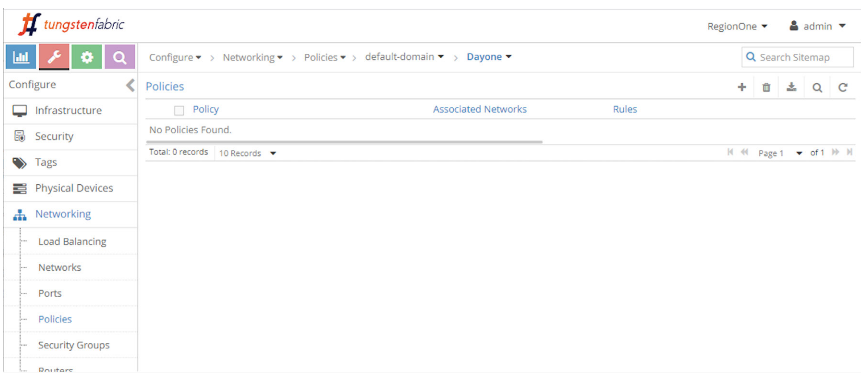


Figure 4.32 Configure / Networking / Policies / Default-domain / Dayone

2. Create a new policy by clicking on the add + button . This will pop up a new create policy window (Figure 4.33).

3. Enter the policy name as WEB2APP

NOTE A policy can have one or more rules, which either PASS or DENY traffic based on the tuples mentioned.

4. Click on the add + button on the create window to add a policy rule.

A new rule will require fields like action, protocol, source, source port, destination, and destination port along with options like log, services, mirror and QoS:

- Action field can either be PASS and DENY.
- Protocol field can be either of the list:
  - Any (Any protocol)
  - TCP
  - UDP
  - ICMP
  - ICMP6
- Source address field can be:
  - Specific IP address
  - CIDR
  - VN from the VN list
  - Another policy
  - Security Group
- Source port field can be simple port number, range of ports or the keyword ANY.
- Destination address field will be similar to source address field.
- Destination port will be similar to source port field.
- Log check box will enable or disable logging of messages when a particular rule is hit during policy checks.
- Services check is used to create service chains. We will discuss more about this option in the service chaining section of the book.
- Mirror option is used to enable mirroring of traffic matching a particular rule of the policy.
- QoS is used to enable marking of traffic matching the rule of the policy.

5. Select the protocol as ICMP.
6. Select the source network as VNWeb from the list

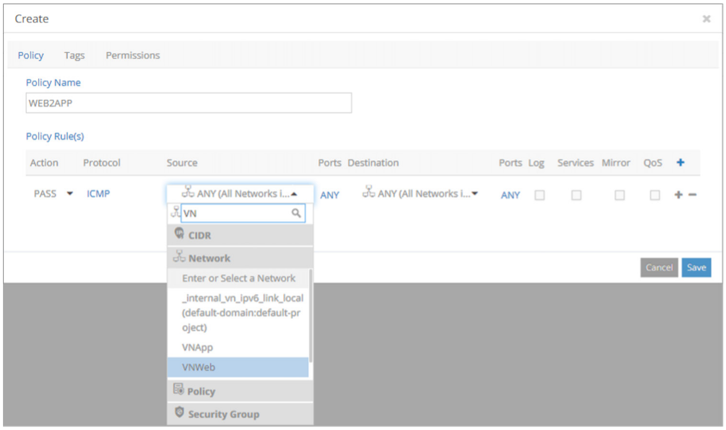


Figure 4.33 Policy Screen

7. Leave the source port as ANY.
8. Select the destination network as VNApp.
9. Click Save to create the policy.

The Policy window should look something like Figure 4.34 after successfully creating the policy.

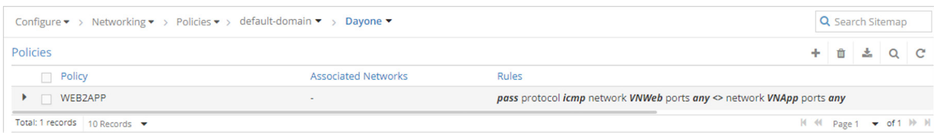


Figure 4.34 Policy Screen with a Policy

Checking Communication Between WebServer1 and AppServer1

In previous sections, we created one VM in VNWeb and one in VNApp virtual networks. In this section, let's try to communicate between them using our trusted ping packet.

1. Log in to Horizon GUI.
2. Make sure the project selected on the project drop down list is Dayone.
3. Expand the Project menu on the left-hand side.

4. Click on Compute.
5. Click on Instances. After successfully loading the page, it should look like Figure 4.35.

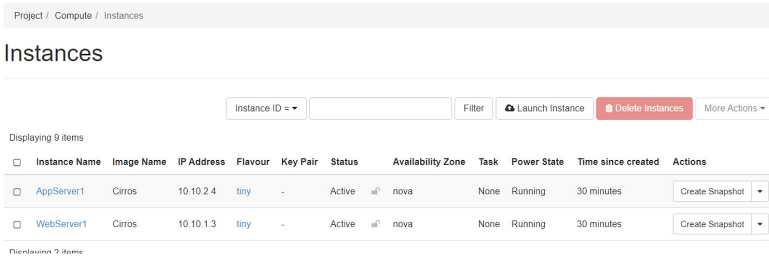


Figure 4.35 Accessing Instances

6. Click on WebServer1. This will navigate to the WebServer1 window with tabs for the virtual machine, check console logs, virtual console to the VM, and action logs.
7. Click on Console tab.
8. Clicking on the grey bar on the top of tab will enable keystrokes on this console.
9. Try pinging the APPServer1 (Figure 4.36). As of now, the ping should fail.

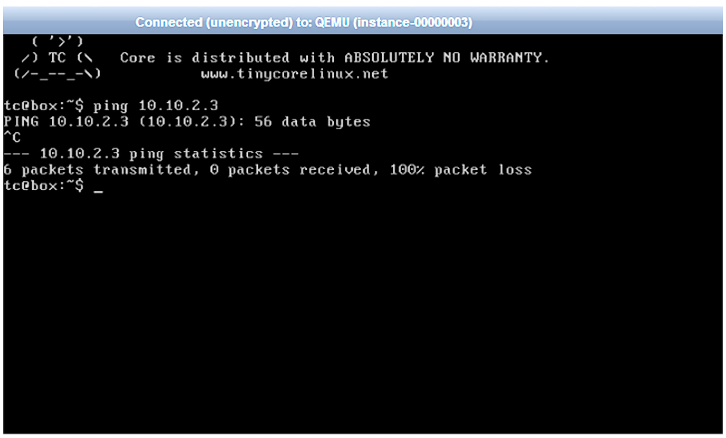


Figure 4.36 Ping from VM

Attaching Policy to Virtual Networks

Any policy created and not applied will not have the desired outcome, i.e. allowing two virtual networks to communicate or deny a certain type of traffic between a VM can be achieved only after applying the policy to the desired object.

In our example, we want to establish communication between WebServer1 and APPServer1, which belongs to VNWeb and VNApp. Based on this requirement, let's apply the policy to both VNs.

1. Navigate to Configure / Networking / Networks in Contrail GUI.

NOTE Make sure that your Dayone project is selected in the project navigation section of the window.

2. Click on the gear icon against the virtual network VNWeb and click on Edit.
3. This will load the Edit Network screen. It will be similar to the VN creation screen.
4. Click in the empty text box under Network Policy. This will enable a list of the policies existing in the system. As of now, there is only one policy created which enables communication between VNWeb and VNApp.
5. Select the policy. The text box should now be populated as in Figure 4.37.

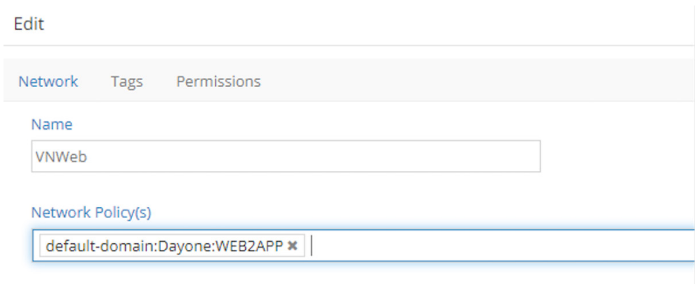


Figure 4.37 Assigning Policy to a Network

6. Click Save on the edit screen.
7. After applying the policy to VNWeb, the screen should like Figure 4.38.

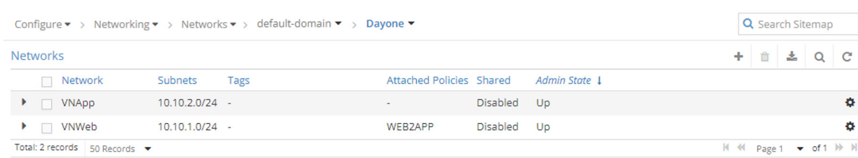


Figure 4.38 VNWeb Applied with Policy

8. Repeat the procedure from step 2 to step 7 for applying the policy to VNApp VN.
9. Once completed, both VNs should exhibit the attached policies section with policy name (Figure 4.39).

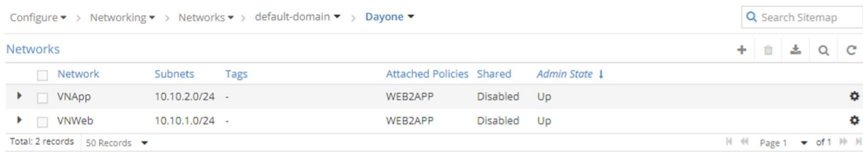


Figure 4.39 VNWeb Applied with Policy

Verifying Communication Between WebServer1 and AppServer1 After Applying the Policy to VNs

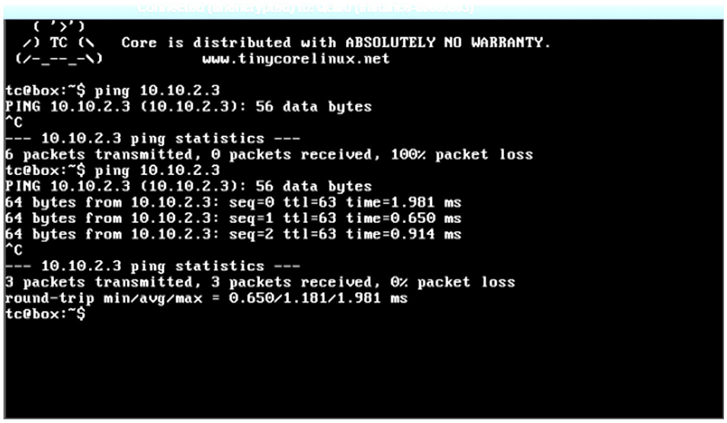


Figure 4.40 Ping between VMs After Applying Policy

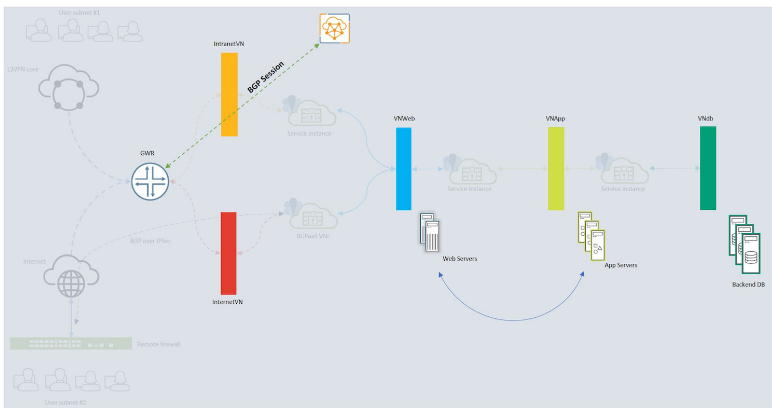


Figure 4.41 Three-tier Application Creation Progress After Allowing Communication Between VNWeb and VNApp Through Policies

Verifying Communication to VNWeb From WAN

Service Chaining

Service chaining is the ability of an SDN controller to automate the process of chaining network services such as L4-L7 firewalls, NAT, and IPS/IDS between VNs. Traffic from a VN can go through an arbitrary graph of service nodes before reaching another VN. The traffic can take different paths based on a service (for example, an IDS can redirect traffic to a DPI engine or it can be replicated for monitoring purposes). This is exactly the same as what is achieved in traditional networking by connecting the cables in a way that traffic from one segment flows to another through a firewall, load balancer or IDS/IPS, or a switch port that is configured in analyzer mode.

For example, an application hosted on a web server should be accessed from the internet only after passing through a stateful firewall. To achieve this, the connection from the internet will be connected to the firewall WAN port and the web server segment will be connected to another port of the firewall that can be the DMZ (Figuyre 4.42).

The firewall will also have one port for out-of-band management. This management port does not have anything to do with traffic transiting from the internet to DMZ or vice-versa.

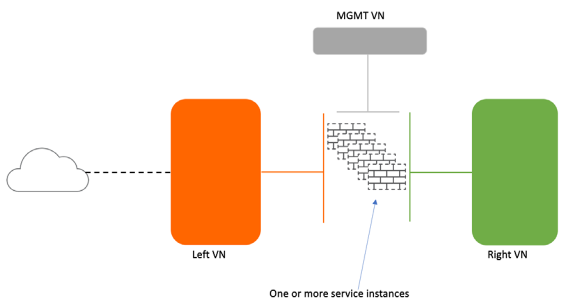


Figure 4.42 Service Chain Example

Contrail also offers horizontal scaling of service chains (*scaling-out*) where one can add more than one VNF instance to load balance traffic across virtual networks passing through a SI. This is useful in cases where one instance of the VNF is not sufficient to handle the volume of data passing through it. Contrail will treat each instance as one ECMP path and load balance the traffic across them.

Adding More Than One Type of SI To the Path

In many use cases, it may be necessary to add more than one type of VNF between the path of VNs. This type of service chaining is referred to as *complex service chaining*. As an example, you can have the firewall and then the load balancer VNF placed one after another between VNs, as shown in Figure 4.43.

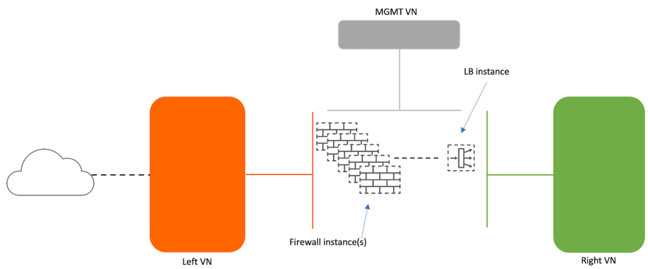


Figure 4.43 Service Chain with More Than One Type of VNF in the Path

Configuration Objects Used in SI Creation

The following configuration resources and objects need to be in place before adding them to the service instance:

- Virtual Networks: These need to be connected through a service instance.
  - Left VN
  - Right VN
  - Mgmt VN
- Virtual machine(s) for service instance: This is an instance created using a VNF image. Typically required to have a minimum of three interfaces: mgmt, left, and right.
- Service template: This defines the template for the type of service instance to be created. In this template we define the type of SI to be created:
  - In-network
  - In-network NAT
  - Transparent

- Service instance: This is the configuration object where you map VMIs of the VNFs to the network segment types defined in the SI template.
- Service policy: This is used to place the SI between the networks mentioned in source and destination of the rule.

Service Chain Types

- In-network service chaining: This type of service chaining provides gateway service where packets are routed between VNs. Routes from VNs connected of the left and right interfaces or service instance are leaked to routing tables on the other side
- In-network NAT: Similar to in-network service chain. However, route leaking is done only in one direction
- Transparent: Also referred to as bump in the wire. Does not modify packet. Used in IDP/IPS or L2 firewall requirements.

Configuring a Service Chain Between VNs

In this lab, we'll create a service chain between Intranet VN and VNWeb as shown in Figure 4.44.

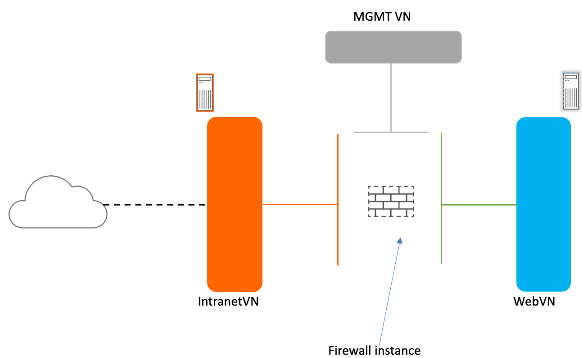


Figure 4.44 Simple In-network Service Chain

What we already have in our lab topology is an IntranetVN, VNWeb, and VMs in it and routing between IntranetVN and DCGW, as shown in Figure 4.45.

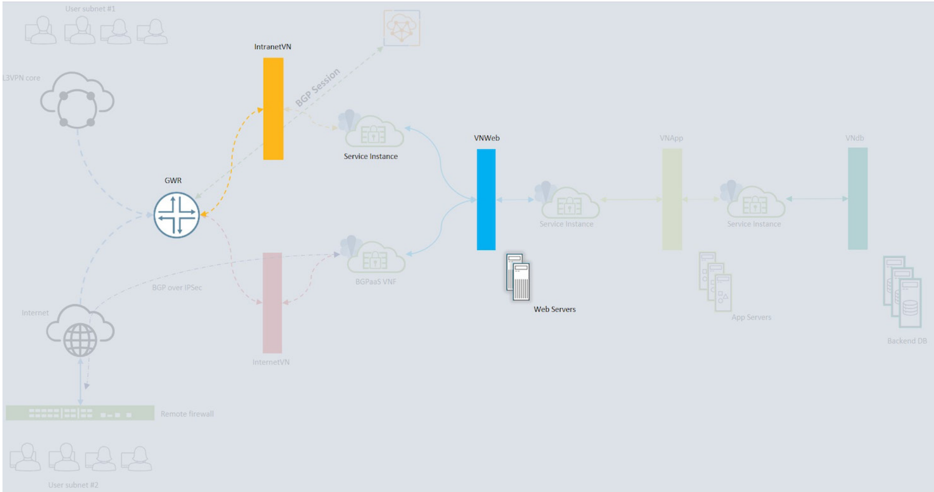


Figure 4.45 *Understanding the Requirement of Service Chaining*

Create a Service Chain

- One VM in Intranet VM (only for testing traffic between VNs).
- Spin up a vSRX instance. (You can download the vSRX image from the Juniper software download site.)
- Create a service template with three interfaces, mgmt, left, and right.
- Create a service instance configuration object using the service template.
- Map the vSRX instance's interfaces to service instance interfaces.
- Create a policy between IntranetVN and VNWeb which calls the SI.
- Apply the policy to both VNs.

After completing these steps, you should be able to communicate between the VMs of IntranetVN and VNWeb and be able to see the routes for the VNWeb in the routing table of DCGW.

First launch the VMs using the steps mentioned previously, replacing the following values while creating the VMs:

1. VM name with TESTVM.
2. Network with IntranetVN.

3. Create another VM with the following parameters.
4. VM name as vSRX-intra-web-1.
5. Select vSRX in the image.
6. Select Medium in the flavor.
7. Three networks in the order of Management, IntranetVN, and VNWeb .

Create a service template

1. Open the Contrail GUI.
2. Navigate to Services / Service Templates (see Figure 4.46).
3. Click on + (Add) to create a new template.
4. Enter the name as firewall.
5. Keep the version to default value (v2).
6. Choose virtual machine asVirtualization Type.
7. Select the Service Mode as “In-Network”.
8. Select the Service Type as “Firewall”.
9. Click on the + icon three times for interfaces: management, left, and right.
10. Click on Save.

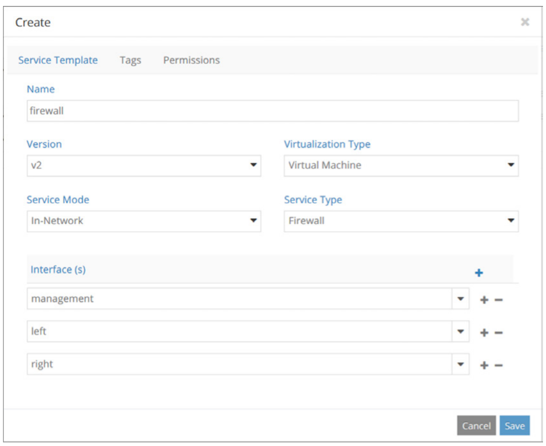


Figure 4.46 Service Template Creation

Ceate a Service Instance (see Figures 4.47 and 4.48):

1. Navigate to Configure / Services and click on "Service Instances".
2. Click on + in Service Instances screen.
3. Enter the name as FWSI-Intra-web in create screen.
4. Select "firewall – [in-network (management, left, right)]" in the Service Template drop down list.
5. Select VMMgmt for management, IntranetVN for left, and VNWeb for right, in the Interface Type to Virtual Network mapping list.
6. Expand port tuples.
7. Click on + (add) under port tuples. This will add a new list of port tuples.
8. Expand this new list by clicking on the > (expand) icon.
9. Select VMIs of the vSRX instance and click on Save.

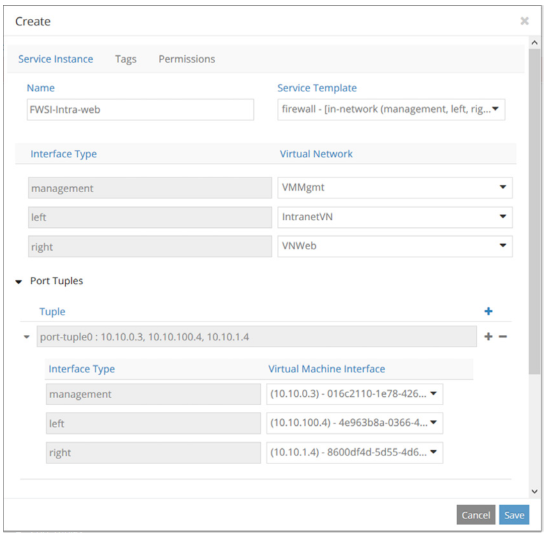


Figure 4.47 Service Instance Create Screen

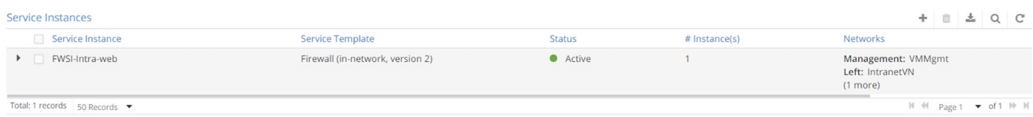


Figure 4.48 Service Instances Screen After First SI is Created

Create a policy to allow traffic between IntranetVN and VNWeb and apply FWSI-Intra-web as the SI within it (see Figures 4.49 and 4.50):

1. Navigate to Configure / Networking.
2. Click on Policies.
3. Click on + (add) to create a new policy.
4. Enter the policy name as Intra2web.
5. Click on + (add) to add new policy rule in the Create screen.
6. Set the action as Pass.
7. Select the source and destination networks as IntranetVN and VNWeb, respectively.
8. Leave the source and destination ports as default.
9. Enable the Services check box. This enables a new text box below the rules section which allows you to select service instances to be applied for this rule.
10. Click in the Service Instances text box and click on FWSI-Intra-web.
11. Click on Save.

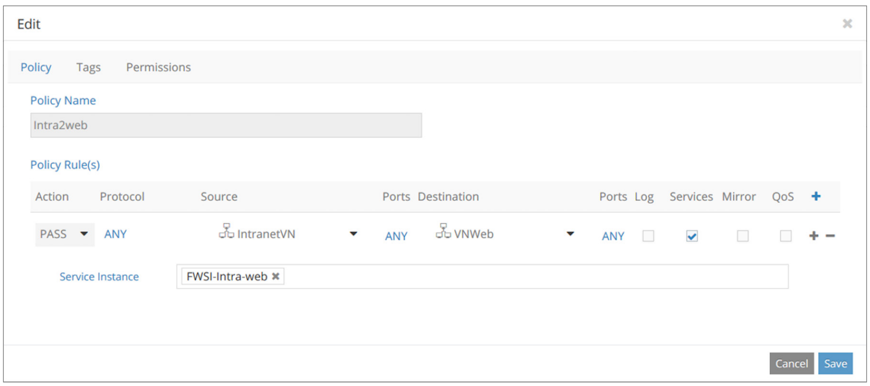


Figure 4.49 SI Policy Create Screen

Figure 4.50 Policy With SI Enabled

So far, we have created the service instance and referenced it in a policy. However, we have yet to apply it to the networks that need to communicate through the service instance. Let's take a moment and verify the routing tables on the controller, compute, and DCGW before applying SI policy.

Routing Table on DCGW

```
root@DCGW1> show route table Intranet.inet.0

Intranet.inet.0: 4 destinations, 7 routes (3 active, 0 holddown, 2 hidden)
+ = Active Route, - = Last Active, * = Both

0.0.0.0/0          *[Static/5] 20:43:41
                    > via lt-0/0/0.1
10.10.100.3/32     *[BGP/170] 01:03:20, MED 100, localpref 200, from 192.168.110.55 << Test VM IP
                      AS path: ?, validation-state: unverified
                    > via gr-0/0/0.32771, Push 25
                    [BGP/170] 01:03:20, MED 100, localpref 200, from 192.168.110.56
                      AS path: ?, validation-state: unverified
                    > via gr-0/0/0.32771, Push 25
10.10.100.4/32    *[BGP/170] 01:00:41, MED 100, localpref 200, from 192.168.110.55  << SRX Interface
IP
                      AS path: ?, validation-state: unverified
                    > via gr-0/0/0.32771, Push 44
                    [BGP/170] 01:00:41, MED 100, localpref 200, from 192.168.110.56
                      AS path: ?, validation-state: unverified
                    > via gr-0/0/0.32771, Push 44
```

If you observe the routing table, you will realize that the Intranet.inet.0 does not yet have the routes from VNWeb

Routing Table on Computes

To verify the routing tables on computes, you need to find the compute that hosts the service instance or TESTVM.

You can find the compute host the VMs by navigating to the Horizon Dashboard / Project / Compute / Instances and clicking on the instance whose compute you wish to determine. For this purpose, let's assume that the VMs related to Intranet-VM are hosted on compute2.

Execute the `vif --list` command to list the interfaces with IP address from IntranetVN(10.10.100.0/24):

```
[root@compute2 ~]# vif --list | grep -C2 10.10.100

vif0/4      OS: tap683dedd7-dd NH: 31
            Type:Virtual HWaddr:00:00:5e:00:01:00 IPaddr:10.10.100.3
            Vrf:3 Mcast Vrf:3 Flags:PL3L2DEr QOS:-1 Ref:6
            RX packets:518  bytes:22356 errors:0
--
```

```
vif0/6      OS: tap4e963b8a-03 NH: 59
            Type:Virtual HWaddr:00:00:5e:00:01:00 IPaddr:10.10.100.4
            Vrf:3 Mcast Vrf:3 Flags:PL3L2DEr QOS:-1 Ref:6
            RX packets:513  bytes:22089 errors:0
```

From the output you can see that the IntranetVN on this compute is mapped to vrf 3. List the routes from the vrf 3 routing table:

```
root@compute2 ~]# docker exec -ti vrouter_vrouter-agent_1 rt --dump 3 | grep 10.10.100.
10.10.100.0/32         24            TF          -              1        -
10.10.100.1/32         32            PT          -             13        -
10.10.100.2/32         32            PT          -             13        -
10.10.100.3/32         32             P          -             31        2:68:3d:ed:d7:dd(114544)
10.10.100.4/32         32             P          -             59        2:4e:96:3b:8a:3(212900)
10.10.100.5/32         24            TF          -              1        -
10.10.100.6/32         24            TF          -              1        -
--snip-

[root@compute2 ~]# docker exec -ti -e COLUMNS=$COLUMNS -e LINES=$LINES vrouter_vrouter-
agent_1 rt --dump 3 | egrep -E "10.10.1.3"
10.10.103.0/24          0            LP      299872              38        -
10.10.113.0/24          0            LP      299872              38        -
10.10.123.0/24          0            LP      299872              38        -
10.10.133.0/24          0            LP      299872              38        -
10.10.143.0/24          0            LP      299872              38        -
10.10.153.0/24          0            LP      299872              38        -
10.10.163.0/24          0            LP      299872              38        -
10.10.173.0/24          0            LP      299872              38        -
10.10.183.0/24          0            LP      299872              38        -
10.10.193.0/24          0            LP      299872              38        -
```

Here we are trying to find the route for 10.10.1.3 of VNWeb in the IntranetVN. As the service instance is not applied yet, the route for this IP is still missing.

Apply the newly created policy to the networks (see Figure 4.51):

1. Navigate to Networks.
2. Edit Intranet VN.
3. Select the policy "default-domain:Dayone:Intra2web" under the text box "Network Policy(s)".
4. Click on Save.
5. Repeat the above steps for VNWeb.

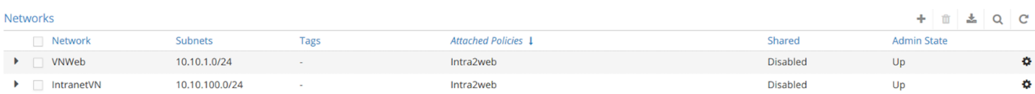


Figure 4.51 Networks After Applying Policy

Verifying the Service Chain

Check the DCGW routing table:

```
root@DCGW1> show route table Intranet.inet.0

Intranet.inet.0: 7 destinations, 13 routes (5 active, 0 holddown, 4 hidden)
+ = Active Route, - = Last Active, * = Both

0.0.0.0/0          *[Static/5] 21:13:02
                    > via lt-0/0/0.1
10.10.1.3/32       *[BGP/170] 00:05:02, MED 100, localpref 200, from 192.168.110.55
                      AS path: ?, validation-state: unverified
                    > via gr-0/0/0.32771, Push 44
                    [BGP/170] 00:05:02, MED 100, localpref 200, from 192.168.110.56
                      AS path: ?, validation-state: unverified
                    > via gr-0/0/0.32771, Push 44
10.10.1.4/32       *[BGP/170] 00:05:02, MED 100, localpref 200, from 192.168.110.55
                      AS path: ?, validation-state: unverified
                    > via gr-0/0/0.32771, Push 44
                    [BGP/170] 00:05:02, MED 100, localpref 200, from 192.168.110.56
                      AS path: ?, validation-state: unverified
                    > via gr-0/0/0.32771, Push 44
10.10.100.3/32     *[BGP/170] 01:32:41, MED 100, localpref 200, from 192.168.110.55
                      AS path: ?, validation-state: unverified
                    > via gr-0/0/0.32771, Push 25
                    [BGP/170] 01:32:41, MED 100, localpref 200, from 192.168.110.56
                      AS path: ?, validation-state: unverified
                    > via gr-0/0/0.32771, Push 25
10.10.100.4/32     *[BGP/170] 01:30:02, MED 100, localpref 200, from 192.168.110.55
                      AS path: ?, validation-state: unverified
                    > via gr-0/0/0.32771, Push 44
                    [BGP/170] 01:30:02, MED 100, localpref 200, from 192.168.110.56
                      AS path: ?, validation-state: unverified
                    > via gr-0/0/0.32771, Push 44
```

As the type of service instance is *in-network*, routes from the right side VN are automatically leaked into the left VN, which is the Intranet VN. The IntranetVN will apply the same attributes as its own and advertise it to DCGW:

```
docker exec -ti -e COLUMNS=$COLUMNS -e LINES=$LINES vrouter_vrouter-agent_1 rt --dump 3 | egrep
"10.10.1.3"
10.10.1.3/32           32             P           -              59         -
```

Once the policy with SI is applied to the VNs, the route for the VMs in VNWeb can be seen in the VRF mapped to the IntranetVN. Navigate to Monitor/ Infrastructure / Control Nodes / controller1.example.net and select the IntranetVN from Routing Instance drop down list. Click on Search (see Figure 4.52).

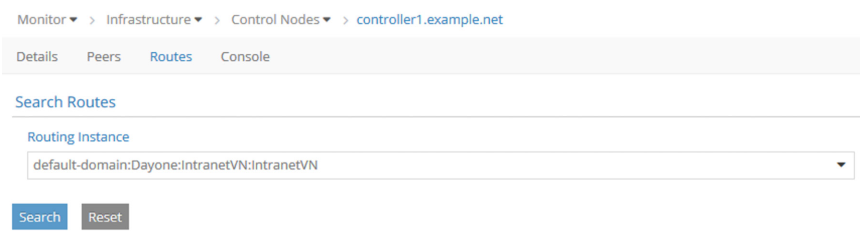
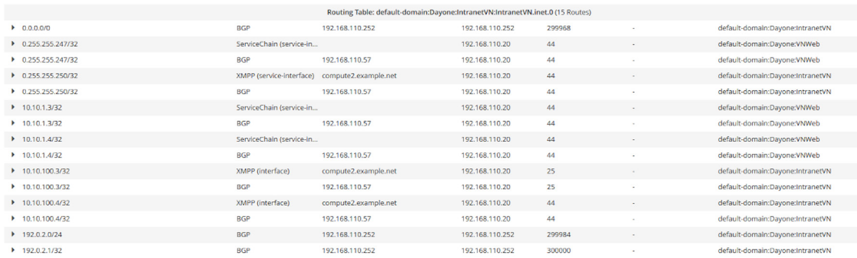


Figure 4.52 *Search Routes*

Scroll down to the section (see Figure 4.53) which displays inet.0 routing tables for the RI:



Routing Table: default-domain:Dayone:IntranetVN:IntranetVN.inet.0 (15 Routes)

0.0.0.0/0	BGP	192.168.110.252	192.168.110.252	299968	-	default-domain:Dayone:IntranetVN
0.255.255.247/32	ServiceChain (service-in...		192.168.110.20	44	-	default-domain:Dayone:VNWeb
0.255.255.247/32	BGP	192.168.110.57	192.168.110.20	44	-	default-domain:Dayone:VNWeb
0.255.255.250/32	XMPP (service-interface)	compute2.example.net	192.168.110.20	44	-	default-domain:Dayone:IntranetVN
0.255.255.250/32	BGP	192.168.110.57	192.168.110.20	44	-	default-domain:Dayone:IntranetVN
10.10.1.3/32	ServiceChain (service-in...		192.168.110.20	44	-	default-domain:Dayone:VNWeb
10.10.1.3/32	BGP	192.168.110.57	192.168.110.20	44	-	default-domain:Dayone:VNWeb
10.10.1.4/32	ServiceChain (service-in...		192.168.110.20	44	-	default-domain:Dayone:VNWeb
10.10.1.4/32	BGP	192.168.110.57	192.168.110.20	44	-	default-domain:Dayone:VNWeb
10.10.100.3/32	XMPP (interface)	compute2.example.net	192.168.110.20	25	-	default-domain:Dayone:IntranetVN
10.10.100.3/32	BGP	192.168.110.57	192.168.110.20	25	-	default-domain:Dayone:IntranetVN
10.10.100.4/32	XMPP (interface)	compute2.example.net	192.168.110.20	44	-	default-domain:Dayone:IntranetVN
10.10.100.4/32	BGP	192.168.110.57	192.168.110.20	44	-	default-domain:Dayone:IntranetVN
192.0.2.0/24	BGP	192.168.110.252	192.168.110.252	299984	-	default-domain:Dayone:IntranetVN
192.0.2.1/32	BGP	192.168.110.252	192.168.110.252	300000	-	default-domain:Dayone:IntranetVN

Figure 4.53 *IntranetVN Routing Table After Service Chaining*

You can see in Figure 4.53 that the routes from 10.10.1.0/24 subnets are leaked to the IntranetVN routing table and its original VN as default-domain:Dayone:VNWeb. The second column indicates the protocol as Service-Chain. The fourth and fifth columns are the next hop IP and label used to reach these destinations.

Let's select the next hop and label of one IP address of a VM from the VNWeb and check the CLI outputs on that compute:

- Prefix: 10.10.1.3/32
- Protocol: ServiceChain
- Source: <blank>
- Next Hop: 192.168.110.20
- Label: 44
- Origin VN: default-domain:Dayone:VNWeb

```
[root@compute2 ~]# docker exec -ti vrouter_vrouter-agent_1 nh --get 44
Id:44         Type:Encap          Fmly: AF_INET  Rid:0  Ref_cnt:2          Vrf:2
              Flags:Valid, Multicast, Etree Root,
              EncapFmly:0806 Oif:3 Len:14
              Encap Data: 02 b2 97 f6 e3 f9 00 00 5e 00 01 00 08 00

[root@compute2 ~]# docker exec -ti vrouter_vrouter-agent_1 vif --get 3
Vrouter Interface Table

Flags: P=Policy, X=Cross Connect, S=Service Chain, Mr=Receive Mirror
       Mt=Transmit Mirror, Tc=Transmit Checksum Offload, L3=Layer 3, L2=Layer 2
       D=DHCP, Vp=Vhost Physical, Pr=Promiscuous, Vnt=Native Vlan Tagged
      Mnp=No MAC Proxy, Dpdk=DPDK PMD Interface, Rfl=Receive Filtering Offload, Mon=Interface is 
Monitored
      Uuf=Unknown Unicast Flood, Vof=VLAN insert/strip offload, Df=Drop New Flows, L=MAC Learning 
Enabled
      Proxy=MAC Requests Proxied Always, Er=Etree Root, Mn=Mirror without Vlan Tag, HbsL=HBS Left Intf
      HbsR=HBS Right Intf, Ig=Igmp Trap Enabled

vif0/3      OS: tapb297f6e3-f9 NH: 37
            Type:Virtual HWaddr:00:00:5e:00:01:00 IPaddr:10.10.1.3
            Vrf:2 Mcast Vrf:2 Flags:PL3L2DEr QOS:-1 Ref:6
            RX packets:23322  bytes:982028 errors:0
            TX packets:23588  bytes:993202 errors:0
            Drops:0
```

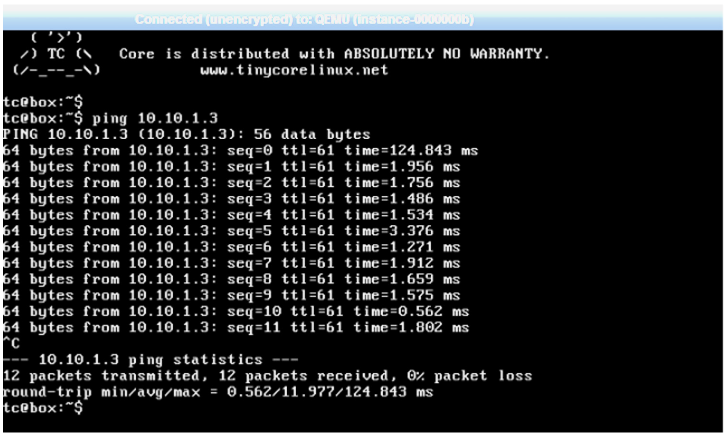


Figure 4.54 Ping from IntranetVN Test VM to VM in VNWeb

Now let's initiate a ping from the test VM we had created in IntranetVN towards the VM in *VNWeb* (see Figure 4.55) and you can also see the three-tier application process in Figure 4.56.

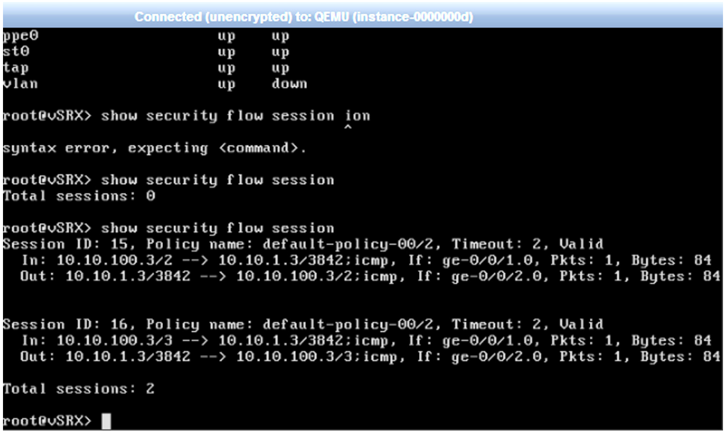


Figure 4.55 Flow Sessions On vSRX During Ping Test

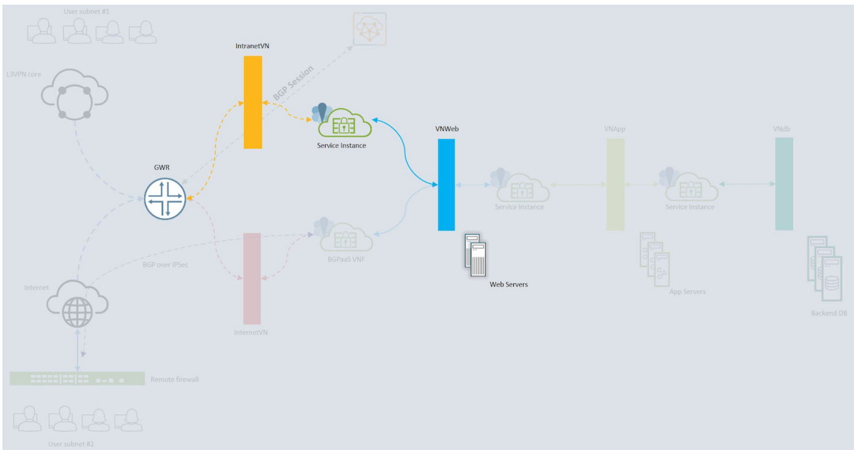


Figure 4.56 Three-tier Application Progress After Completing Service Chain Configuration

Scaling Service Chain Horizontally

In a typical network scenario, one would increase the bandwidth capacity of a network function by either adding more cards or replacing the device with a bigger device. This is *vertical scaling*.

The other way of achieving increased capacity is by adding more devices and instances horizontally and load balancing the traffic across these devices. We refer this to as *horizontal scaling* or *scale-out*. Contrail networking supports horizontal scaling a service chain by adding more VNF instances to the SI. In such service chains, all VNFs will be in active-active state.

To Add More VNFs to SI (Figures 4.57 and 4.58)

1. Create a VM as before in the Openstack GUI and note down the IP addresses for this VM(s). (It will be required while selecting the interfaces during port tuple addition.)
2. Navigate to the Contrail GUI Configure / Services / Service Instances.
3. Click the gear icon for newly created SI "FWSI-Intra-web" and click on Edit.t
4. Expand port tuples by click the > icon.
5. Click on (add) + under Port Tuples to add interfaces to the newly created FW.
6. Expand the added port tuple.
7. Select the Interfaces for the VM created.
8. Click Save.

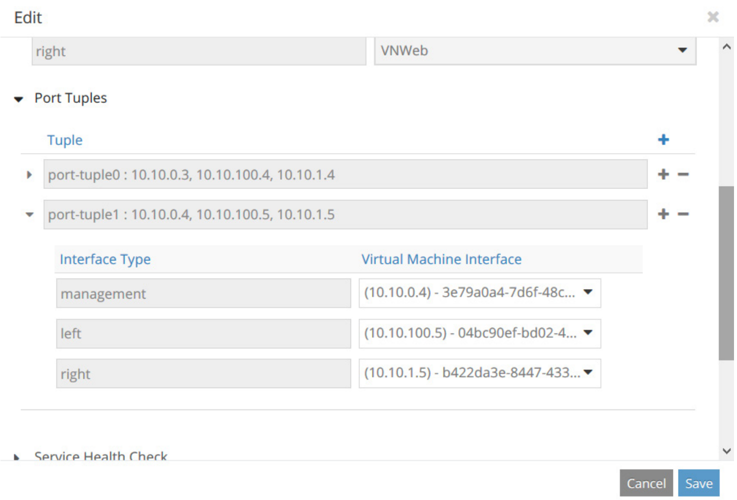


Figure 4.57 Adding Port Tuples to Existing SI

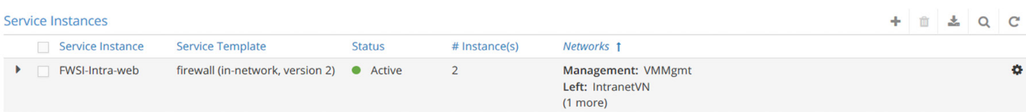


Figure 4.58 Service Instances After Adding New Instance

Verify Service Chain Scaling

Now let's verify DCGW:

```
root@DCGW1> show route table Intranet.inet.0

Intranet.inet.0: 11 destinations, 29 routes (9 active, 0 holddown, 8 hidden)
+ = Active Route, - = Last Active, * = Both

0.0.0.0/0          *[Static/5] 1w4d 16:23:14
                    > via lt-0/0/0.1
10.10.1.3/32       *[BGP/170] 00:03:14, MED 100, localpref 200, from 192.168.110.55
                      AS path: ?, validation-state: unverified
                    > via gr-0/0/0.32771, Push 62
                    [BGP/170] 1w2d 08:00:49, MED 100, localpref 200, from 192.168.110.55
                      AS path: ?, validation-state: unverified
                    > via gr-0/0/0.32769, Push 44
                    [BGP/170] 00:03:14, MED 100, localpref 200, from 192.168.110.57
                      AS path: ?, validation-state: unverified
                    > via gr-0/0/0.32771, Push 62
                    [BGP/170] 1w2d 08:00:52, MED 100, localpref 200, from 192.168.110.57
                      AS path: ?, validation-state: unverified
                    > via gr-0/0/0.32769, Push 44
10.10.1.4/32       *[BGP/170] 00:03:14, MED 100, localpref 200, from 192.168.110.55
                      AS path: ?, validation-state: unverified
                    > via gr-0/0/0.32771, Push 62
                    [BGP/170] 1w2d 08:00:49, MED 100, localpref 200, from 192.168.110.55
                      AS path: ?, validation-state: unverified
                    > via gr-0/0/0.32769, Push 44
                    [BGP/170] 00:03:14, MED 100, localpref 200, from 192.168.110.57
                      AS path: ?, validation-state: unverified
                    > via gr-0/0/0.32771, Push 62
                    [BGP/170] 1w2d 08:00:52, MED 100, localpref 200, from 192.168.110.57
                      AS path: ?, validation-state: unverified
                    > via gr-0/0/0.32769, Push 44
10.10.1.5/32       *[BGP/170] 00:03:14, MED 100, localpref 200, from 192.168.110.55
                      AS path: ?, validation-state: unverified
                    > via gr-0/0/0.32771, Push 62
                    [BGP/170] 00:31:28, MED 100, localpref 200, from 192.168.110.55
                      AS path: ?, validation-state: unverified
                    > via gr-0/0/0.32769, Push 44
                    [BGP/170] 00:03:14, MED 100, localpref 200, from 192.168.110.57
                      AS path: ?, validation-state: unverified
                    > via gr-0/0/0.32771, Push 62
                    [BGP/170] 00:31:28, MED 100, localpref 200, from 192.168.110.57
                      AS path: ?, validation-state: unverified
                    > via gr-0/0/0.32769, Push 44
10.10.100.3/32     *[BGP/170] 1w2d 08:00:49, MED 100, localpref 200, from 192.168.110.55
                      AS path: ?, validation-state: unverified
                    > via gr-0/0/0.32769, Push 25
                    [BGP/170] 1w2d 08:00:52, MED 100, localpref 200, from 192.168.110.57
                      AS path: ?, validation-state: unverified
                    > via gr-0/0/0.32769, Push 25
10.10.100.4/32     *[BGP/170] 1w2d 08:00:49, MED 100, localpref 200, from 192.168.110.55
                      AS path: ?, validation-state: unverified
                    > via gr-0/0/0.32769, Push 44
                    [BGP/170] 1w2d 08:00:52, MED 100, localpref 200, from 192.168.110.57
                      AS path: ?, validation-state: unverified
```

```
                > via gr-0/0/0.32769, Push 44
10.10.100.5/32     *[BGP/170] 00:30:12, MED 100, localpref 200, from 192.168.110.55
                      AS path: ?, validation-state: unverified
                    > via gr-0/0/0.32771, Push 62
                    [BGP/170] 00:30:12, MED 100, localpref 200, from 192.168.110.57
                      AS path: ?, validation-state: unverified
                    > via gr-0/0/0.32771, Push 62
192.0.2.0/24       *[Direct/0] 1w3d 18:53:38
                    > via lo0.1
192.0.2.1/32       *[Local/0] 1w3d 18:53:38
                      Local via lo0.1

root@DCGW1>
root@DCGW1> show interfaces gr-0/0/0.32771
  Logical interface gr-0/0/0.32771 (Index 336) (SNMP ifIndex 562)
   Flags: Up Point-To-Point SNMP-Traps 0x0 IP-Header 192.168.110.19:192.168.110.252:47:df:64:00000800
00000000 Encapsulation: GRE-NULL
    Gre keepalives configured: Off, Gre keepalives adjacency state: down
    Input packets : 0
    Output packets: 0
    Protocol inet, MTU: 1476
      Flags: None
    Protocol mpls, MTU: 1464, Maximum labels: 3
      Flags: None

root@DCGW1> show interfaces gr-0/0/0.32769
  Logical interface gr-0/0/0.32769 (Index 334) (SNMP ifIndex 559)
    Flags: Up Point-To-Point SNMP-Traps 0x0 IP-
Header 192.168.110.20:192.168.110.252:47:df:64:0000080000000000 Encapsulation: GRE-NULL
    Gre keepalives configured: Off, Gre keepalives adjacency state: down
    Input packets : 0
    Output packets: 0
    Protocol inet, MTU: 1476
      Flags: None
    Protocol mpls, MTU: 1464, Maximum labels: 3
      Flags: Is-Primary

root@DCGW1>
```

On compute1

```
[root@compute1 ~]# docker exec -it 0a5e4862832b mpls --get 62
MPLS Input Label Map

   Label    NextHop
-------------------
      62       88
[root@compute1 ~]# docker exec -it 0a5e4862832b nh --get 88
Id:88         Type:Encap          Fmly: AF_INET  Rid:0  Ref_cnt:5          Vrf:10
              Flags:Valid, Policy, Etree Root,
              EncapFmly:0806 Oif:9 Len:14
              Encap Data: 02 04 bc 90 ef bd 00 00 5e 00 01 00 08 00

[root@compute1 ~]# docker exec -it 0a5e4862832b vif --get 9
Vrouter Interface Table
```

```
Flags: P=Policy, X=Cross Connect, S=Service Chain, Mr=Receive Mirror
      Mt=Transmit Mirror, Tc=Transmit Checksum Offload, L3=Layer 3, L2=Layer 2
      D=DHCP, Vp=Vhost Physical, Pr=Promiscuous, Vnt=Native Vlan Tagged
      Mnp=No MAC Proxy, Dpdk=DPDK PMD Interface, Rfl=Receive Filtering Offload, Mon=Interface is
Monitored
      Uuf=Unknown Unicast Flood, Vof=VLAN insert/strip offload, Df=Drop New Flows, L=MAC Learning
Enabled
      Proxy=MAC Requests Proxied Always, Er=Etree Root, Mn=Mirror without Vlan Tag, HbsL=HBS Left Intf
      HbsR=HBS Right Intf, Ig=Igmp Trap Enabled

vif0/9      OS: tap04bc90ef-bd NH: 88
            Type:Virtual HWaddr:00:00:5e:00:01:00 IPaddr:10.10.100.5
            Vrf:10 Mcast Vrf:10 Flags:PL3L2DEr QOS:-1 Ref:6
            RX packets:314  bytes:13731 errors:0
            TX packets:730  bytes:31256 errors:0
            Drops:6
```

And on compute2

```
[root@compute2 ~]# docker exec -it 711f9aac1834 mpls --get 44
MPLS Input Label Map

   Label    NextHop
-------------------
      44       59
[root@compute2 ~]# docker exec -it 711f9aac1834 nh --get 59
Id:59         Type:Encap          Fmly: AF_INET  Rid:0  Ref_cnt:5          Vrf:3
              Flags:Valid, Policy, Etree Root,
              EncapFmly:0806 Oif:6 Len:14
              Encap Data: 02 4e 96 3b 8a 03 00 00 5e 00 01 00 08 00

[root@compute2 ~]# docker exec -it 711f9aac1834 vif --get 6
Vrouter Interface Table

Flags: P=Policy, X=Cross Connect, S=Service Chain, Mr=Receive Mirror
       Mt=Transmit Mirror, Tc=Transmit Checksum Offload, L3=Layer 3, L2=Layer 2
       D=DHCP, Vp=Vhost Physical, Pr=Promiscuous, Vnt=Native Vlan Tagged
      Mnp=No MAC Proxy, Dpdk=DPDK PMD Interface, Rfl=Receive Filtering Offload, Mon=Interface is
Monitored
      Uuf=Unknown Unicast Flood, Vof=VLAN insert/strip offload, Df=Drop New Flows, L=MAC Learning
Enabled
      Proxy=MAC Requests Proxied Always, Er=Etree Root, Mn=Mirror without Vlan Tag, HbsL=HBS Left Intf
      HbsR=HBS Right Intf, Ig=Igmp Trap Enabled

vif0/6      OS: tap4e963b8a-03 NH: 59
            Type:Virtual HWaddr:00:00:5e:00:01:00 IPaddr:10.10.100.4
            Vrf:3 Mcast Vrf:3 Flags:PL3L2DEr QOS:-1 Ref:6
            RX packets:95528  bytes:4014623 errors:0
            TX packets:190121  bytes:7987600 errors:0
            Drops:6

[root@compute2 ~]#

[root@compute2 ~]# docker exec -it 711f9aac1834 rt --get 10.10.1.3/32 --vrf 3 --family inet
Match 10.10.1.3/32 in vRouter inet4 table 0/3/unicast
```

```
Flags: L=Label Valid, P=Proxy ARP, T=Trap ARP, F=Flood ARP
vRouter inet4 routing table 0/3/unicast
Destination           PPL        Flags        Label         Nexthop    Stitched MAC(Index)
10.10.1.3/32            0            P          -             94       -
[root@compute2 ~]#
[root@compute2 ~]#
[root@compute2 ~]# docker exec -it 711f9aac1834 nh --get 94
Id:94         Type:Composite      Fmly: AF_INET  Rid:0  Ref_cnt:6          Vrf:3
              Flags:Valid, Policy, Ecmp, Etree Root,
              Valid Hash Key Parameters: Proto,SrcIP,SrcPort,DstIp,DstPort
              Sub NH(label): 60(44) 47(62)

Id:60         Type:Encap          Fmly: AF_INET  Rid:0  Ref_cnt:4          Vrf:3
              Flags:Valid, Etree Root,
              EncapFmly:0806 Oif:6 Len:14
              Encap Data: 02 4e 96 3b 8a 03 00 00 5e 00 01 00 08 00

Id:47         Type:Tunnel         Fmly: AF_INET  Rid:0  Ref_cnt:28         Vrf:0
              Flags:Valid, MPLSoUDP, Etree Root,
              Oif:0 Len:14 Data:f8 f2 1e 79 5b d0 f8 f2 1e 79 66 90 08 00
              Sip:192.168.110.20 Dip:192.168.110.19

[root@compute1 ~]# docker exec -it 0a5e4862832b mpls --get 62
MPLS Input Label Map

   Label    NextHop
-------------------
      62       88
[root@compute1 ~]# docker exec -it 0a5e4862832b nh --get 88
Id:88         Type:Encap          Fmly: AF_INET  Rid:0  Ref_cnt:5          Vrf:10
              Flags:Valid, Policy, Etree Root,
              EncapFmly:0806 Oif:9 Len:14
              Encap Data: 02 04 bc 90 ef bd 00 00 5e 00 01 00 08 00

[root@compute1 ~]# docker exec -it 0a5e4862832b vif --get 9
Vrouter Interface Table

Flags: P=Policy, X=Cross Connect, S=Service Chain, Mr=Receive Mirror
       Mt=Transmit Mirror, Tc=Transmit Checksum Offload, L3=Layer 3, L2=Layer 2
       D=DHCP, Vp=Vhost Physical, Pr=Promiscuous, Vnt=Native Vlan Tagged
      Mnp=No MAC Proxy, Dpdk=DPDK PMD Interface, Rfl=Receive Filtering Offload, Mon=Interface is
Monitored
      Uuf=Unknown Unicast Flood, Vof=VLAN insert/strip offload, Df=Drop New Flows, L=MAC Learning
Enabled
      Proxy=MAC Requests Proxied Always, Er=Etree Root, Mn=Mirror without Vlan Tag, HbsL=HBS Left Intf
      HbsR=HBS Right Intf, Ig=Igmp Trap Enabled

vif0/9      OS: tap04bc90ef-bd NH: 88
            Type:Virtual HWaddr:00:00:5e:00:01:00 IPaddr:10.10.100.5
            Vrf:10 Mcast Vrf:10 Flags:PL3L2DEr QOS:-1 Ref:6
            RX packets:314  bytes:13731 errors:0
            TX packets:730  bytes:31256 errors:0
            Drops:6
```

BGPaaS

BGP-as-a-service is used to exchange routes between a VNF and controller on behalf of a VN. This is usually done when the VNF is receiving routes over a tunnel and these routes are required to be placed in the Contrail routing table.

Based on the example diagram (Figure 4.59), the LeftVN is configured with an RT so that the DCGW can learn it from the Contrail controller in a VRF having reachability to the internet. These routes are advertised further in the service provider network, providing reachability between the remote site firewall public interface and the left interface of the FW instance.

If we configure service chaining, routes from RightVN will be leaked to leftVN and will be advertised further to DCGW. We do not wish to do this route leaking of RightVN to DCGW. Instead, we will configure the firewall at the remote site and firewall instance to form an IPsec tunnel between them. These firewalls will be also be configured to exchange routes over the IPsec tunnel using any routing protocol. Once we have the routes on the firewall instance, we need to advertise them to the Contrail controller so that the packets destined for the remote subnet (192.168.1.0/24) can be forwarded to the left interface of the firewall instance by vRouter.

This can be done by configuring BGP peering between the left interface of the FW instance and the gateway IP address of LeftVN (see Figure 4.59). This is referred to as BGP-as-a-service. BGP packets originated from the FW instance addressed to the LeftVN gateway IP address and the vRouter proxies these packets to the controller.

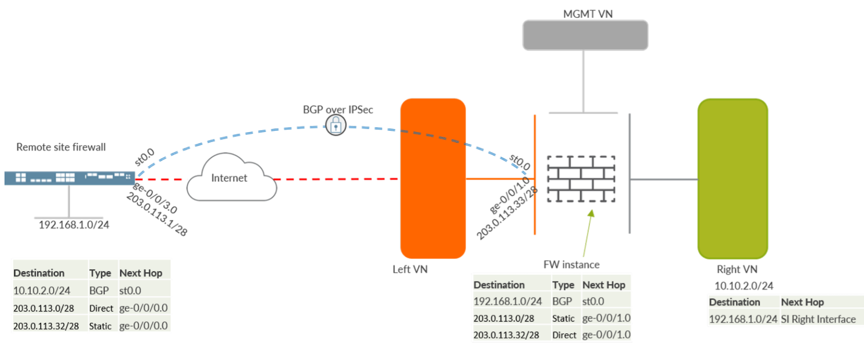


Figure 4.59 BGPaaS Example Topology

BGPaaS Configuration

1. Navigate to Configure / Services and click BGP as a Service (see Figures 4.60 and 4.61).
2. Click on (add) + to create a BGP as a service configuration.
3. Enter the name of the configuration object as VPNFW.
4. Enter the autonomous system ID of the vSRX as 65514.
5. Select the Virtual Machine Interface of the vSRX which is pointing towards VNWeb. In our Dayone example, it is 10.10.1.5.
6. Click on Save.

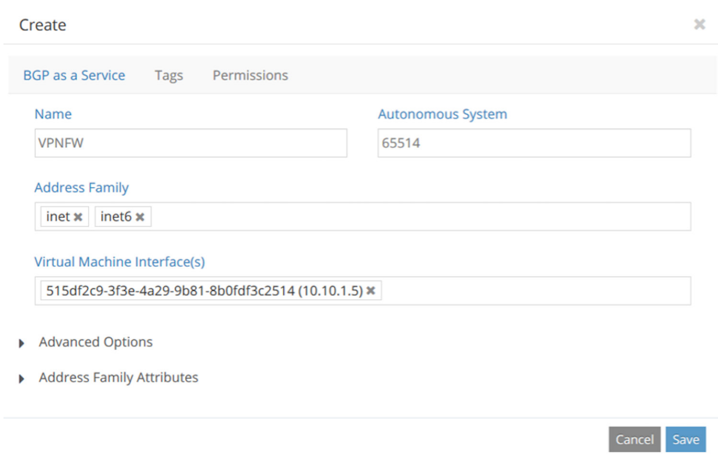


Figure 4.60 BGP as a Service Creation

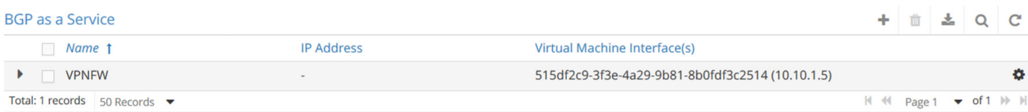


Figure 4.61 BGP as a Service Window

BGPaaS Verification

Once configuration is in place with the vSRX and Contrail controller, verify the state from both sides using the following procedure.

Note that the VMI UUID from Figure 4.61. It is 515df2c9-3f3e-4a29-9b81-8b0fd-f3c2514. We will use this to verify the neighbor status on controller using CLI introspect.

To verify BGP neighbor state and routes received from controller on the vSRX:

1. Access the command prompt of the vSRX.

2. Use the `show bgp summary` command to verify the BGP state as Established.

3. Also check for received routes using the `show route receive-protocol bgp 10.10.1.1` command:

```
root@vSRX> show bgp summary
Groups: 1 Peers: 1 Down peers: 0
Table          Tot Paths  Act Paths Suppressed    History Damp State    Pending
inet.0                 3          2          0          0          0          0
inet6.0                0          0          0          0          0          0
Peer                     AS      InPkt     OutPkt    OutQ   Flaps Last Up/Dwn State|#Active/
Received/Accepted/Damped...
10.10.1.1             64512        202        225       0       1        1:05 Establ
  inet.0: 2/3/3/0
  inet6.0: 0/0/0/0

root@vSRX>
root@vSRX> show route receive-protocol bgp 10.10.1.1

inet.0: 9 destinations, 10 routes (9 active, 0 holddown, 0 hidden)
  Prefix                  Nexthop              MED     Lclpref    AS path
* 10.10.1.3/32            10.10.1.1            100                64512 ?
* 10.10.1.4/32            10.10.1.1            100                64512 ?
  10.10.1.5/32            10.10.1.1            100                64512 ?
```

To Verify BGP State On the Controller

1. Access the introspect page of the Contrail controller using the URL: http://<controller_IP>:8083.

The controller introspect page (Figure 4.62) will provide several options to view data.

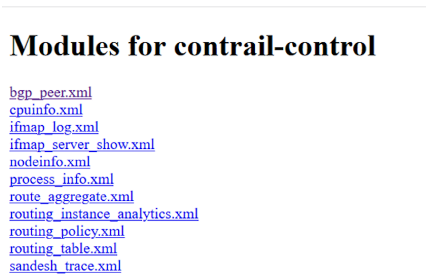


Figure 4.62 Controller Introspect Home Page

2. Click on bgp_peer.xml

The next page will present you with more options for bgp_peer. Click on the Send button.

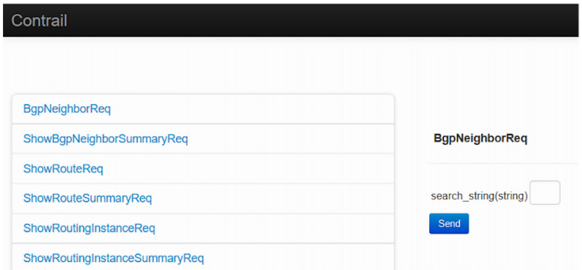


Figure 4.63 Controller BGP_Peer Introspect Options Page

Figure 4.64 shows the vSRX as a neighbor in the neighbor list of the controller.

Figure 4.64 BGPaaS Neighbor in BGP Neighbor List

Verify the BGP Status on Controller Using CLI Introspect

Information required:

- The URL to query can be found from the introspect page: *http://10.254.0.55:8083/Snh_BgpNeighborReq?search_string*=.
- VThe MI UUID of BGPaaS peer can be found in the BGPaaS screen. The UUID here is *515df2c9-3f3e-4a29-9b81-8b0fdf3c2514*.

1. Access the shell prompt of either controller or any other machine which has reachability to the controller.

2. Run the following:

```
[root@CONTROLLER1 ~]# curl http://10.254.0.55:8083/Snh_BgpNeighborReq?search_string=515df2c9-3f3e-
4a29-9b81-8b0fdf3c2514 | python -c 'import sys;import xml.dom.minidom;s=sys.stdin.read();print xml.
dom.minidom.parseString(s).toprettyxml()'
  % Total     % Received % Xferd  Average Speed   Time    Time     Time  Current
```

```
                                 Dload  Upload   Total   Spent    Left  Speed
100  8112  100  8112    0     0  1620k      0 --:--:-- --:--:-- --:--:-- 1980k
<?xml version="1.0" ?>
<?xml-stylesheet type="text/xsl" href="/universal_parse.xsl"?>
<BgpNeighborListResp type="sandesh">
        <neighbors identifier="1" type="list">
                <list size="1" type="struct">
                        <BgpNeighborResp>
                                <instance_name identifier="53" type="string">default-
domain:Dayone:VNWeb:VNWeb</instance_name>
                          <peer identifier="1" link="BgpNeighborReq" type="string">515df2c9-3f3e-
4a29-9b81-8b0fdf3c2514</peer>
                          <deleted identifier="36" type="bool">false</deleted>
                          <peer_address identifier="2" link="BgpNeighborReq"
type="string">10.10.1.5</peer_address>
                                <peer_id identifier="25" type="string">10.10.0.3</peer_id>
                                <peer_asn identifier="3" type="u32">65514</peer_asn>
                                <encoding identifier="6" type="string">BGP</encoding>
                                <peer_type identifier="7" type="string">external</peer_type>
                                <state identifier="8" type="string">Established</state>
                                <closed_at identifier="43" type="string"/>
```

The portion after the (pipe) | is added to the command to pretty up the XML output for legibility.

Based on the output, the BGP peer is established. This complete output can be used in the same way as the Junos CLI to troubleshoot neighbor-related issues.

Checking Routes on the Controller

1. Navigate to the routes page on the controller (see Figure 4.65).
2. Select the routing instance as VNWeb.
3. Enter three octets of remote end prefix in the Prefix text box and an asterisk (*) for fourth octet.
4. Click on Search.

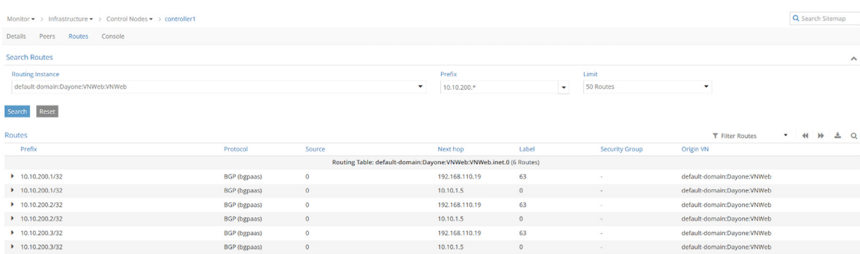


Figure 4.65 Routing Table on the Controller

And the finished three-tier application topology looks like Figure 4.66.

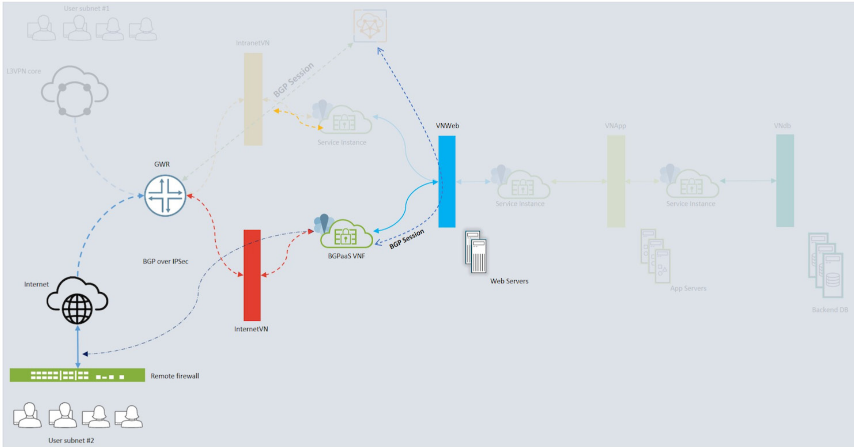


Figure 4.66 Three-tier Application Progress After Completing BGPaaS Configuration

MORE? Remember the Juniper TechLibrary supports Contrail Networking with an entire suite of excellent documentation: https://www.juniper.net/documentation/product/en_US/contrail-networking/19/.