Contrail Networking Architecture Guide

Detailed Technical Description of the Contrail Virtual Networking and Security Platform

November 2019
Juniper Networks, Inc.
1133 Innovation Way
Sunnyvale, California 94089
USA
408-745-2000
www.juniper.net

Juniper Networks, the Juniper Networks logo, Juniper, and Junos are registered trademarks of Juniper Networks, Inc. and/or its affiliates in the United States and other countries. All other trademarks may be property of their respective owners.
Juniper Networks assumes no responsibility for any inaccuracies in this document. Juniper Networks reserves the right to change, modify, transfer, or otherwise revise this publication without notice.
The information in this document is current as of the date on the title page.

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

Copyright © 2019 Juniper Networks, Inc. All rights reserved.
The information in this document is current as of the date on the title page.

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

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

**Introduction** ................................................................................................................................................ 5  
**Use Cases** .................................................................................................................................................. 5  
**Key Features of Contrail Networking for Virtual Machines and Containers** ........................................... 6  

**How Contrail Networking Works** ................................................................................................................... 7  
**Contrail Networking Working with An Orchestrator** .................................................................................... 7  
**Interaction With An Orchestrator** .................................................................................................................. 9  
**Architecture Details of vRouter** .................................................................................................................... 11  
**Detailed Packet Processing Logic In a vRouter** ............................................................................................. 13  
**Packet Flow Between VMs In The Same Subnet** ............................................................................................ 15  
**Packet Flow Between VMs In Different Subnets** ........................................................................................... 16  

**Service Chains** ........................................................................................................................................... 16  
**Basic Service Chain** ................................................................................................................................... 18  
**Scaled-out Services** ..................................................................................................................................... 18  
**Policy-based Steering** ................................................................................................................................... 19  
**Active-Standby Service Chains** .................................................................................................................... 19  

**Application-based Security Policies** ............................................................................................................ 19  
**Deployment Options for vRouter** ................................................................................................................ 25  
**Kernel Module vRouter** ............................................................................................................................. 26  
**DPDK vRouter** ............................................................................................................................................. 27  
**SR-IOV (Single Root – Input/Output Virtualization)** .................................................................................... 27  
**Smart NIC vRouter** ...................................................................................................................................... 27  

**Contrail Controller Microservices** ................................................................................................................. 27  
**OpenStack Orchestration with Contrail Networking** .................................................................................... 28  
**Kubernetes Container Orchestration with Contrail Networking** ................................................................ 30  
**Contrail Networking and VMware vCenter** .................................................................................................. 34  

**Nested Kubernetes with OpenStack or vCenter** .......................................................................................... 36  
**Connecting to Physical Networks** ............................................................................................................. 37  
**BGP-Enabled Gateway** ............................................................................................................................... 37  
**Source NAT** .................................................................................................................................................. 39  
**Routing in Underlay** .................................................................................................................................... 39  

**Fabric Management in Contrail Networking** ............................................................................................... 39  
**Scope of Fabric Management** ...................................................................................................................... 39
Key Concepts Used in Fabric Lifecycle Management

Roles
Namespaces
Virtual Port Groups

Procedure for Fabric Creation

Fabric Creation
Device Discovery
Role Assignment
Autoconfiguration
Device Operations

Lifecycle Management and Virtual Networking for Bare Metal Servers

Virtual Networking for Physical Servers

Packets Between Servers in the Same Virtual Network
Packets Between Servers in Different Networks
Traffic Between a Physical Server and a VM in the Same Virtual Network
Traffic Between a Physical Server and a VM in Different Virtual Networks

Physical Network Configuration for VMware vCenter

CVFM Design Overview
Introduction

This document describes how Contrail Networking provides a scalable virtual networking platform that works with a variety of virtual machine and container orchestrators, and can integrate with physical networking and compute infrastructure. This allows users to take advantage of open-source orchestration while preserving existing infrastructure, procedures, and workloads, which mitigates disruption and cost.

As virtualization becomes a key technology for delivery of both public and private cloud services, issues of network scale are becoming apparent with the virtualization technologies that have been in widespread use to date (E.g. VMware with L2 networking, and OpenStack with stock Nova, Neutron or ML2 networking). Contrail Networking provides a highly scalable virtual networking platform that is designed to support multitenant networks in the largest environments while supporting multiple orchestrators simultaneously.

Since there are very few datacenter deployments that are truly “greenfield”, there are nearly always requirements to integrate workloads deployed on new infrastructure with workloads and networks that have been previously deployed. This document describes a set of scenarios for deployments where new cloud infrastructure will be deployed, and where coexistence with existing infrastructure is also needed.

Use Cases

The following common use cases are covered in this document:

- Enable Platform-as-a-Service and Software-as-a-Service with high scalability and flexibility in OpenStack-managed datacenters
- Virtual networking with container management systems such as Kubernetes, including with Red Hat OpenShift
- Allow an existing virtualized environment running VMware vCenter to use Contrail Networking virtual networking between virtual machines
- Connect Contrail Networking virtual networks to physical networks using gateway routers with BGP peering
- Provide lifecycle management for devices in a IP fabric, including zero-touch provisioning, base configuration, underlay configuration, and overlay configuration
- Provide lifecycle management for bare metal servers, including OS provisioning and attachment into virtual networks by configuring VXLAN VTEPs in connected switches

These use cases can be deployed in any combination to address the specific requirements in a variety of deployment scenarios. Figure 1, below, illustrates the main feature areas of Contrail Networking.
The key feature areas that enable support of the main use cases are:

- Virtual networking using encapsulation tunnels between virtualized hosts
- Plugins for open-source orchestrators for virtual machines and containers
- Application-based security policies based on tags
- Integration with VMware orchestration stack
- Lifecycle management for switches and routers in data center fabrics

Since the same controller and forwarding components are used in these use cases, Contrail Networking can provide a consistent interface for managing connectivity in all the environments it supports, and is able provide seamless connectivity between workloads managed by different orchestulators, whether virtual machines, containers, or bare metal servers, and to destinations in external networks.

**Key Features of Contrail Networking for Virtual Machines and Containers**

Contrail Networking manages and implements virtual networking in cloud environments using OpenStack, Kubernetes and VMware orchestrators. Contrail Networking uses overlay networks between vRouters that run on each host. It is based on proven, standards-based networking technologies that today support the wide-area networks of the world’s major service providers, but repurposed to work in data centers with virtualized workloads and cloud automation. It provides many enhanced features over the native networking implementations of orchestrators, including:

- Highly scalable, multitenant networking
- Multitenant IP address management
- DHCP, ARP proxies to avoid flooding into networks
• Local DNS resolution
• Distributed firewall with access control lists
• Application-based security policies
• Distributed load balancing across hosts
• Network address translation (1:1 floating IPs and N:1 SNAT)
• Service chaining with virtual network functions
• Dual stack IPv4 and IPv6
• BGP peering with gateway routers

The following sections describe in detail how the controller interacts with an orchestrator and the vRouters, and how the above features are implemented and configured in each vRouter.

Fabric and bare metal server management with Contrail Networking is described in the section Fabric Management in Contrail Networking, later in this document.

How Contrail Networking Works

This section describes the software architecture of the Contrail Networking controller and of the vRouter, which forwards packets in each host, and describes the interaction between vRouters and the Contrail Networking controller when virtual machines or containers are started and then exchange packets with each other.

Contrail Networking Working with An Orchestrator

Contrail Networking consists of two primary pieces of software:

• Contrail Networking Controller – a set of software services that maintains a model of networks and network policies, typically running on several servers for high availability
• Contrail Networking vRouter – installed in each virtualized host to enforce network and security policies, and to perform packet forwarding

A typical deployment of Contrail Networking is shown in Figure 2, below.
The Contrail Networking controller is integrated with a cloud management system such as OpenStack or Kubernetes, and its function is to ensure that when a virtual machine (VM) or container is created, it has network connectivity according to the network policies specified in the controller or orchestrator.

The Contrail Networking controller integrates with the orchestrator via a software plugin that implements the networking service of the orchestrator. For instance, the Contrail Networking plugin for OpenStack implements the Neutron API, and the `kube-network-manager` and `CNI` (container network interface) components listen to network-related events using the Kubernetes (K8s) API.

The Contrail Networking vRouter replaces Linux bridge and the iptables utility, or Open vSwitch networking, on the compute hosts, and the controller configures the vRouters to implement the desired networking and security policies.

Packets from a VM on one host that have a destination running on a different host are encapsulated in MPLS over UDP, MPLS over GRE, or VXLAN where the destination of the outer header is the IP address of the host that the destination VM is running on. The controller is responsible for installing the set of routes in each VRF of each vRouter that implements network policies. E.g. by default, VMs in the same network can communicate with each other, but not with VMs in different networks, unless
this is specifically enabled in a network policy. Communication between the controller and vRouters is via XMPP, a widely used and flexible messaging protocol.

A key feature of cloud automation is that users can request resources for their applications without needing to understand details of how or even where resources will be provided. This is normally done via a portal that presents a set of service offerings from which a user can select, and which get translated into API calls into underlying systems including the cloud orchestrator to spin up virtual machines or containers with the necessary memory, disk, and CPU capacity for the user’s requirements. Service offerings can be as simple as a VM with specific memory, disk, and CPU allocated to it, or may include an entire application stack composed of multiple pre-configured software instances.

**Interaction With An Orchestrator**

The architecture of the Contrail Networking controller and vRouter, and the interaction with an orchestrator, is shown in Figure 3, below.

![Figure 3: Interaction between an orchestrator, Contrail Networking Controller and Contrail Networking vRouter](image-url)
How Contrail Networking Works

Each interface of VMs running on the host is connected to a VRF that contains the forwarding tables for the corresponding network that contains the IP address of that interface. A vRouter only has VRFs for networks that have interfaces in them on that host, including the Fabric VRF that connects to the physical interface of the host. Contrail Networking virtual networking uses encapsulation tunneling to transport packets between VMs on different hosts, and the encapsulation and decapsulation happens between the Fabric VRF and the VM VRFs. This is explained in more detail in the next section.

When a new virtual workload is created, an event is seen in the plugin and sent into the controller, which then sends requests to the agent for routes to be installed in the VRFs for virtual networks, and the agent then configures them in the forwarder.

The logical flow for configuring networking on a new VM with a single interface is as follows:

1. Networks and network policies are defined in either the orchestrator or Contrail Networking using UI, CLI, or REST API. A network is primarily defined as a pool of IP addresses which will be allocated to interfaces when VMs are created.

2. VM is requested to be launched by a user of the orchestrator, including which network its interface is in.

3. The orchestrator selects a host for the new VM to run on, and instructs the compute agent on that host to fetch its image and start the VM.

4. The Contrail Networking plugin receives events or API calls from the networking service of the orchestrator instructing it to set up the networking for the interface of the new VM that will be started. These instructions are converted into Contrail Networking REST calls and sent to the Contrail Networking controller.

5. The Contrail Networking controller sends a request to the vRouter agent for the new VM virtual interface to be connected to the specified virtual network. The vRouter agent instructs the vRouter Forwarder to connect the VM interface to the VRF for the virtual network. The VRF is created, if not present, and the interface is connected to it.

6. The compute agent starts the VM which will usually be configured to request IP addresses for each of its interfaces using DHCP. The vRouter proxies the DHCP requests and responds with the interface IP, default gateway, and DNS server addresses.

7. Once the interface is active and has an IP address from DHCP, the vRouter will install routes to the VM’s IP and MAC addresses with a next hop of the VM virtual interface.

8. The vRouter assigns a label for the interface and installs a label route in the MPLS table. The vRouter sends an XMPP message to the controller containing a route to the new VM. The route has a next hop of the IP address of the server that the vRouter is
running on, and specifies an encapsulation protocol using the label that was just allocated.

9. The controller distributes the route to the new VM to the other vRouters with VMs in the same network and in other networks, as allowed by network policy.

10. The controller sends routes for the other VMs, as allowed by policy, to the vRouter of the new VM.

At the end of this procedure, the routes in the VRFs of all the vRouters in the data center have been updated to implement the configured network policies, taking account of the new VM.

**Architecture Details of vRouter**

This section describes the architecture of the Contrail Networking vRouter in more detail. A conceptual view of the functional components of the Contrail Networking vRouter is shown in Figure 4, below.
How Contrail Networking Works

Figure 4: OpenStack and Contrail Networking agents on a compute node

The vRouter agent runs in the user space of the host operating system, while the forwarder can run as a kernel module, in user space when DPDK is used, or can run in a programmable network interface card, also known as a “smart NIC”. These options are described in more detail in the section Deployment Options for vRouter. The more commonly used kernel module option is illustrated here.

The agent maintains a session with the controller and is sent information about VRFs, routes, and access control lists (ACLs) that it needs. The agent stores the information in its own database and uses the information to configure the forwarder. Interfaces get connected into VRFs, and the forwarding information base (FIB) in each VRF is configured with forwarding entries.

Each VRF has its own forwarding and flow tables, while the MPLS and VXLAN tables are global within the vRouter. The forwarding tables contain routes for both the IP and MAC addresses of destinations and the IP-to-MAC association is used to provide proxy ARP capability. The values of
labels in the MPLS table are selected by the vRouter when VM interfaces come up, and are only locally significant to that vRouter. The VXLAN network identifiers are global across all the VRFs of the same virtual network in different vRouters within a Contrail Networking domain.

**Detailed Packet Processing Logic In a vRouter**
The logic details for packets flowing from a VM, and into a VM, are slightly different and described in Figure 5 and Figure 6.

*Figure 5: Logic for a packet arriving in a vRouter from a VM interface*
When a packet is sent from a VM through a virtual interface, it is received by the forwarder, which first checks if there is an entry matching the packets’ 5-tuple (protocol, source and destination IP addresses, source and destination TCP or UDP ports) in the flow table of the VRF that the interface is in. There won’t be an entry if this is the first packet in a flow, and the forwarder sends the packet to the agent over the pkt0 interface. The agent determines the action for the flow based on the VRF routing table and access control list, and updates the flow table with the result. The actions can be DROP, FORWARD or NAT. If the packet is to be forwarded, the forwarder checks to see if the destination MAC address is its own MAC address, which will be the case if the VM is sending a packet to the default gateway when the destination is outside the VM’s subnet. In that case, the next hop for destination is looked up in the IP forwarding table, otherwise the MAC address is used for lookup.

![Diagram](image)

Figure 6: Logic for a packet arriving in a vRouter from the physical network
When a packet arrives from the physical network, the vRouter first checks if the packet has a supported encapsulation or not. If not, the packet is sent to the host operating system. For MPLS over UDP and MPLS over GRE, the label identifies the VM interface directly, but VXLAN requires that the destination MAC address in the inner header be looked up in the VRF identified by the VXLAN network identifier (VNI). Once the interface is identified, the vRouter can forward the packet immediately if there is no policy flag set for the interface (indicating that all protocols and all TCP/UDP ports are permitted). Otherwise the 5-tuple is used to look up the flow in the flow table and the same logic as described for an outgoing packet is used.

**Packet Flow Between VMs In The Same Subnet**

The sequence of action that occurs when a VM first sends a packet to another VM is shown in the following diagram. The starting point is that both VMs have booted and the controller has sent L2 (MAC) and L3 (IP) routes to both vRouters to enable communication between the VMs.

**Figure 7: Sequence when a VM sends a packet to another VM**

1. VM1 needs to send a packet to VM2, so first looks up its own DNS cache for the IP address, but since this is the first packet, there is no entry.
2. VM1 sends a DNS request to the DNS server address that was supplied in the DHCP response when its interface came up.
3. The vRouter traps the DNS request and forwards it to the DNS server running in the Contrail Networking controller.
4. The DNS server in the controller responds with the IP address of VM2.
5. The vRouter sends the DNS response to VM1.
6. VM1 needs to form an Ethernet frame, so needs the MAC address for VM2. It checks its own ARP cache, but there is no entry, since this is the first packet.
7. VM1 sends out an ARP request.
8. The vRouter traps the ARP request and looks up the MAC address for IP-VM2 in its own forwarding tables and find the association in the L2/L3 routes that the controller sent it for VM2.
9. The vRouter sends an ARP reply to VM1 with the MAC address of VM2.
10. A TCP timeout occurs in the network stack of VM1.
11. The network stack of VM1 retries sending the packet, and this time finds the MAC address of VM2 in the ARP cache and can form an Ethernet frame and send it out.
12. The vRouter looks up the MAC address for VM2 and finds an encapsulation route. The vRouter builds the outer header and sends the resulting packet to S2.
13. The vRouter on S2 decapsulates the packet and looks up the MPLS label to identify the virtual interface to send the original Ethernet frame into. The Ethernet frame is sent into the interface and received by VM2.

Packet Flow Between VMs In Different Subnets

The sequence when sending packets to destinations in a different subnet is identical except that VM1 will send the packet in an Ethernet frame with the MAC address of the default gateway, whose IP address was supplied in the DHCP response that the vRouter supplied when VM1 booted. When VM1 does an ARP request for the gateway IP address, the vRouter responds with its own MAC address. When VM1 sends an Ethernet frame using that gateway MAC address, the vRouter uses the destination IP address of the packet inside the frame to look up the forwarding table in the VRF to find a route, which will be via an encapsulation tunnel to the host that the destination VM is running on.

Service Chains

A service chain is formed when a network policy specifies that traffic between two networks has to flow through one or more network services, also termed Virtual Network Functions (VNF). The network services are implemented in VMs—identified in Contrail Networking as services—which are then included in policies. Contrail Networking supports service chains in both OpenStack and vCenter environments. The concept of service chaining between two VMs is shown in Figure 8.
When a VM is configured in the controller to be a service instance (VNF), and the service is included in a network policy that is applied to networks the policy refers to, the controller installs routes in the VRFs of the “Left” and “Right” interfaces of the VNF that direct traffic through the VNF. When encapsulation routes are advertised by the VNF vRouter back to the controller, the routes are distributed to other vRouters that have Red and Green VRFs and the end result is a set of routes that direct traffic flowing between the Red and Green networks to pass through the service instance. The labels “Left” and “Right” are used to identify interfaces based on the order that they become active when the VNF is booted. The VNF must have a configuration that will process packets appropriately based on the interfaces that they will arrive on.

As implemented in Contrail Networking, service chain routes are installed in special VRFs that, for clarity, are not shown here, but the principle is the same.
Various service chain scenarios are illustrated in Figure 9, below, and a brief explanation of each follows.

**Basic Service Chain**

In the first panel, a simple service chain has been created by editing the network policy between the Red and Green networks to include the services FW and DPI. These are VMs that were previously started in OpenStack or vCenter and then configured in Contrail Networking to be service instances with interfaces in the Red and Green networks. When the policy is saved and is applied to the two networks, the routes in all the vRouters with Red or Green VMs attached are modified to send traffic between the two networks via the service chain. For instance, prior to modifying the policy, each VRF in the Red network would have had a route to each VM in the Green network with a next hop of the host where the VM is running and a label that was specified by the host vRouter and sent by the controller. The route is modified to have a next hop of the ingress VRF of the FW service instance, and the label that was specified for the FW left interface. The VRF with the right FW interface will contain routes for all Green destinations that point to the left interface of DPI, and the VRF with the right DPI interface will contain routes for all Green destinations with next hop of the host where they are running and the original label. Routing for traffic in the reverse direction is similarly handled.

**Scaled-out Services**

When a single VM does not have the capacity to handle the traffic requirements of a service chain, multiple VMs of the same type can be included in a service, as shown in the second panel. When this
is done, traffic is load-balanced using ECMP across the ingress interfaces of the service chain at both ends, and is also load-balanced between layers of the chain.

New service instances can be added as needed in Contrail Networking, and although the ECMP hash algorithm would normally move most sessions to other paths when the number of targets changes, in Contrail Networking this only happens for new flows, since the paths for existing flows are determined from the flow tables described in the section *Detailed Packet Processing Logic In a vRouter*. This behavior is essential for stateful services that must see all packets in a flow, or else the flow will be blocked, resulting in a dropped user session.

The flow tables are also populated to ensure that traffic for the reverse direction in a flow passes through the same service instance that it came from.


**Policy-based Steering**

There are cases where traffic of different types needs to be passed into different services chains. This can be achieved in Contrail Networking by including multiple terms in a network or security policy. In the example in the diagram, traffic on ports 80 and 8080 have to pass through both a firewall (FW-1) and DPI, whereas all other traffic only passes through a firewall (FW-2), which may have a different configuration from FW-1.

**Active-Standby Service Chains**

In some scenarios it is desirable for traffic to normally go through some specific service chain, but if there are issues detected with that chain, then traffic should be switched to a backup. This can be the case where the standby service chain is located in a less favorable geographic location.

Active-standby configuration is achieved in two steps in Contrail Networking. First a route policy is applied to the ingress of each service chain specifying a higher local preference value for the preferred active chain ingress. Secondly, a health check is attached to each chain that can test that service instances are reachable, or that a destination on the other side of the chain can be reached. If the health check fails, then the route to the normally active service chain is withdrawn and traffic will flow through the standby.

**Application-based Security Policies**

Conventional firewall policies contain rules based on individual IP addresses or subnet ranges. In data centers of any size this leads to a proliferation of firewall rules which are difficult to manage when being created and difficult to understand when troubleshooting. This is because the IP address
of a server or VM doesn’t relate to the application, application owner, location, or any other property. For instance, consider an enterprise that has two data centers and deploys a three-tier application in development and production, as shown in Figure 10, below.

![Figure 10: Multiple instances of an application stack requires multiple firewall rules](image)

It is a requirement in this enterprise that the layers of each instance of an application can only communicate with the next layer in the same instance. This requires a separate policy for each of the application instances, as shown. When troubleshooting an issue, the admin must know the relation between IP addresses and application instances, and each time a new instance is deployed, a new firewall rule must be written.

The Contrail Networking controller supports security policies based on tags that can be applied to projects, networks, vRouters, VMs, and interfaces. The tags propagate in the object model to all the objects contained in the object where the tag was applied. Tags have a name and a value. Several tag names are supplied as part of the Contrail Networking distribution. Typical uses for the tag types are shown in the table below:

<table>
<thead>
<tr>
<th>Tag Name</th>
<th>Typical Use</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>application</td>
<td>Identify a group of VMs that run a set of software instances of different types to support service accessed by end-users or other services. Can correspond to a Heat stack.</td>
<td>LAMP stack, Hadoop cluster, set of NTP servers, Openstack/Contrail Networking cluster</td>
</tr>
<tr>
<td>tier</td>
<td>A set of software instances of the same type within an application stack that perform the same function. The number of such instances may be scaled according to performance requirements in different stacks.</td>
<td>Apache web server, Oracle database server, Hadoop slave node, OpenStack service containers</td>
</tr>
<tr>
<td>deployment</td>
<td>Indicates the purpose of a set of VMs. Usually applies to all the VMs in a stack.</td>
<td>development, test, production</td>
</tr>
<tr>
<td>site</td>
<td>Indicates the location of a stack, usually at the granularity of data center.</td>
<td>US East, London, Nevada-2</td>
</tr>
<tr>
<td>custom</td>
<td>New tags can be created as needed.</td>
<td>Instance name</td>
</tr>
<tr>
<td>label</td>
<td>Multiple labels can be applied to provide fine-grained control of data flows within and between stacks.</td>
<td>customer-access, finance-portal, db-client-access</td>
</tr>
</tbody>
</table>

As shown in the table, in addition to the tag types that are provided with Contrail Networking, users can create their own custom tag names as needed, and there is a label type tag which can be used to more finely tune data flows.

Application policies contain rules based on tag values and service groups, which are sets of TCP or UDP port numbers. First the security administrator allocates a tag of type application for the application stack, and then assigns a tag of type tier for each software component of the application. This is illustrated in Figure 11, below.
In this example, the application is tagged *FinancePortal* and the tiers are tagged *web*, *app* and *db*. Service groups have been created for the traffic flows into the application stack and between each layer. The security administrator then creates an application policy, called *Portal-3-Tier* containing rules that will allow just the required traffic flows. An application policy set is then associated with the application tag *FinancePortal* and contains the application policy *Portal-3-Tier*. At this point the application stack can be launched, and the tags applied to the various VMs in the Contrail Networking controller. This causes the controller to calculate which routes need to be sent to each vRouter to enforce the application policy set, and these are then sent to the respective vRouters. If there is one instance of each software component, the routing tables in each vRouter would be as follows:

<table>
<thead>
<tr>
<th>Host</th>
<th>VRF</th>
<th>Source</th>
<th>Destination</th>
<th>Ports</th>
<th>Route</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>Net-web</td>
<td>0.0.0.0/0</td>
<td>10.1.1.3/32</td>
<td>80</td>
<td>Interface for VM-web NH=S2, Lbl=10 Route to Internet</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10.1.1.3/32</td>
<td>10.1.2.3/32</td>
<td>8080</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.0.0.0/0</td>
<td>0.0.0.0/0</td>
<td>Any</td>
<td></td>
</tr>
<tr>
<td>S2</td>
<td>Net-app</td>
<td>10.1.1.3/32</td>
<td>10.1.2.3/32</td>
<td>8080</td>
<td>Interface for VM-app NH=S3, Lbl=12</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10.1.2.3/32</td>
<td>10.1.3.3/32</td>
<td>1521, 1630</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>10.1.2.3/32</td>
<td>10.1.1.3/32</td>
<td>1521, 1630</td>
<td></td>
</tr>
<tr>
<td>S3</td>
<td>Net-db</td>
<td>10.1.2.3/32</td>
<td>10.1.3.3/32</td>
<td>1521, 1630</td>
<td>Interface for VM-db NH=S2, Lbl=12</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10.1.3.3/32</td>
<td>10.1.2.3/32</td>
<td>1521, 1630</td>
<td></td>
</tr>
</tbody>
</table>

The networks and VMs are named here for the tier that they are in. In reality, the relationship between entity names and tiers would not usually be as simple. As can be seen in the table, the routes enable traffic only as specified in the application policy, but here the tag-based rules have been converted into network address-based firewall rules that the vRouter is able to apply.

Having successfully created an application stack, let’s look at what happens when another deployment of the stack is created, as shown in Figure 12, below.
There is nothing in the original policy that prevents traffic flowing from a layer in one deployment into a layer in a different deployment. This behavior can be modified by tagging each component of each stack with a *deployment* tag, and by adding a *match* condition in the application policy to allow traffic to flow between tiers only when the deployment tags match. The updated policy is shown in Figure 13, below.
Now the traffic flows conform to the strict requirements that traffic only flows between components within the same stack.

Applying tags of different types allows the security policies to be applied in multiple dimensions, all in a single policy. For instance, in Figure 14, below, a single policy can segment traffic within individual stacks based on site, but allow sharing of the database tier within a site.

If multiple stacks are deployed within the same combination of sites and deployments, a custom tag for the instance name could be created and a match condition on the instance tag could be used to create the required restriction, as seen in Figure 15, below.
Deployment Options for vRouter

There are several deployment options for vRouter that offer different benefits and ease of use:

- **Kernel Module** – This is the default deployment mode
- **DPDK** – Forwarding acceleration is provided using an Intel library
- **SR-IOV** – Provides direct access to NIC from a VM
- **Smart NIC** – vRouter forwarder is implemented in a programmable NIC

These options are illustrated in Figure 16.

Figure 15: Using a custom tag to segment stacks

The application policy features in Contrail Networking provide a very powerful enforcement framework, while simultaneously enabling dramatic simplification of policies, and reduction in their number.
The features and benefits of each option are described below.

**Kernel Module vRouter**

The default deployment option today is for the vRouter forwarder to be implemented in a module that runs in the KVM kernel, replacing use of the Linux iptables utility or the Open vSwitch forwarding function. Running in the kernel gives the forwarder direct access to network traffic as it passes through the network stack of KVM, and provides significant performance improvement over what can be achieved if the forwarder ran as a process in user space. Among the optimizations that have been implemented are:

- TCP segmentation offload
- Large receive offload
- Use of multi-queue virtio packet processing

The kernel module approach allows users to implement network virtualization using Contrail Networking with minimal dependency on underlying server and NIC hardware. However, only specific kernel versions of KVM are supported, and this is detailed in the release notes for each version of Contrail Networking.
DPDK vRouter

The Data Plane Development Kit (DPDK), from Intel, is a set of libraries and drivers that allow applications running in user space to have direct access to a NIC without going through the KVM network stack. A version of the vRouter forwarder is available that runs in user space and supports DPDK. The DPDK vRouter provides accelerated packet throughput compared to the kernel module with unmodified VMs, and even better performance can be achieved if the guest VMs also have DPDK enabled.

The DPDK vRouter works by dedicating CPU cores to packet forwarding which loop continuously waiting for packets. Not only are these cores not available for running guest VMs, but they always run at 100% CPU utilization, and this can be an issue in some environments.

SR-IOV (Single Root – Input/Output Virtualization)

SR-IOV isn’t strictly a deployment option for vRouter itself, but can be used with vRouter in some applications when maximizing bandwidth is important and dedicating cores for packet forwarding (as in DPDK) is undesirable. SR-IOV allows the hardware resources of a NIC to be shared among multiple clients as if each has sole access, much like a hypervisor does for a CPU. It gives a VM interface direct access to the NIC, so the data path bypasses the hypervisor networking stack, which leads to enhanced performance. SR-IOV can be useful when the VM is performing a gateway function between a physical network and virtual networks, but since SR-IOV involves bypassing the vRouter, the interfaces don’t participate in Contrail Networking virtual networks and don’t participate in network policies and network services.

Smart NIC vRouter

Some new NICs are becoming available which are programmable. The Contrail Networking vRouter forwarder functionality can be implemented on these new NICs, and this brings substantial benefits in performance, particularly for small packet sizes which are dominant in some environments. Additionally, forwarding is almost completely offloaded from the x86 CPU of the server, so cores can be freed up for more VMs.

Contrail Controller Microservices

Newer versions of Contrail Networking (5.0 and later) use a microservices architecture based on Docker containers. The microservices are grouped into “pods”, which themselves are grouped into roles, as shown in Figure 17, below.
The architecture is composable, meaning that each Contrail Networking role and pod can be separately scaled using multiple instances, running on different servers, to support the resilience and performance requirements of a particular deployment.

The layout of Contrail Networking services across servers is controlled by configuration files that are read by the deployment tool, which can be Ansible (using playbooks) or Helm (using charts). Example playbooks and charts are available that cover simple all-in-one deployments where all the services run in the same VM, to high-availability examples involving multiple VMs or bare metal servers.

More details on deployment tools and how to use them can be found on the [Contrail Networking documentation](https://contrail-networking.com) page.

**OpenStack Orchestration with Contrail Networking**

OpenStack is the leading open-source orchestration system for virtual machines and containers. Contrail Networking provides an implementation of OpenStack’s Neutron networking service, and provides many additional capabilities as well. In OpenStack, groups of users are assigned to
“projects” within which resources such as VMs and networks are private and can’t be seen by users in other projects (unless this is specifically enabled). The use of VPNs makes the enforcement of project isolation in the network layer straight-forward, since only routes to allowed destinations are distributed to VRFs in vRouters on compute nodes and no flooding occurs due to the proxy services that vRouter performs.

Earlier in Figure 3, the networking service is Neutron and the compute agent is Nova (the OpenStack compute service).

Contrail Networking can provide seamless networking between VMs and Docker containers when both are deployed in an OpenStack environment.

As shown in Figure 18, below, the Contrail Networking plug-in for OpenStack provides a mapping from the Neutron networking API to Contrail Networking API calls that are performed in the Contrail Networking controller.

Contrail Networking supports definition of networks and subnetworks, plus OpenStack network policies and security groups. These entities can be created in either OpenStack or Contrail Networking and any changes are synchronized between the two systems. Additionally, Contrail Networking supports the OpenStack LBaaS v2 API. However, since Contrail Networking provides a rich superset of networking features over OpenStack, many networking features are only available via the Contrail Networking API and GUI. These include assigning route targets to enable connectivity to external routers, service chaining, configuring BGP route policies, and application policies.
Application security, as described in the section Application-based Security Policies is fully supported when OpenStack uses Contrail Networking. Contrail Networking tags can be applied at the project, network, host, VM, or interface levels, and propagate to be applied to all entities that are contained in the object that a tag is applied to.

Additionally, Contrail Networking supports a set of resources for networking and security that can be controlled using OpenStack Heat templates.

Kubernetes Container Orchestration with Contrail Networking

Containers allow processes to work in isolation from each other, while running on the same host operating system (unlike virtual machines where each VM runs its own complete guest operating system). Applications running in containers will generally start up much faster and perform better than the same application running in a VM, and this is one of the reasons why there is widespread interest in using containers in data centers and for NFV. Docker is a software layer that enables containers to be portable across operating system versions and is used by Kubernetes as a shim layer to manage creation and destruction of containers on servers.

As seen in Figure 19, above, Kubernetes manages groups of containers, that together perform some function, and are called pods. The containers in a pod run on the same server and share an IP address. Sets of identical pods (generally running on different servers) form services and network traffic destined for a service has to be directed to a specific pod within a service. In the default Kubernetes networking implementation, selection of a specific pod is performed either by the

Figure 19: Kubernetes organizes containers into pods and services

As seen in Figure 19, above, Kubernetes manages groups of containers, that together perform some function, and are called pods. The containers in a pod run on the same server and share an IP address. Sets of identical pods (generally running on different servers) form services and network traffic destined for a service has to be directed to a specific pod within a service. In the default Kubernetes networking implementation, selection of a specific pod is performed either by the
application itself using a native Kubernetes API in the sending pod, or, for non-native applications, by a load-balancing proxy using a virtual IP address implemented in Linux iptables on the sending server. The majority of applications are non-native, since they are ports of existing code that was not developed with Kubernetes in mind, and therefore the load-balancing proxy is used.

The standard networking in a Kubernetes environment is effectively flat, with any pod able to communicate with any other pod. Communication from a pod in one namespace (similar to a project in OpenStack) to a pod in another namespace is not prevented if the name of target pod, or its IP address is known. While this model is appropriate in hyper-scale data centers belonging to a single company, it is unsuitable for service providers whose data centers are shared among many end-customers, or in enterprises where traffic for different groups must be isolated from each other.

Contrail Networking virtual networking can be integrated in a Kubernetes environment to provide a range of multitenant networking features in similar fashion as with OpenStack.

This configuration of Contrail Networking with Kubernetes is shown in Figure 20, below.
The architecture for Contrail Networking with Kubernetes orchestration and Docker containers is similar to OpenStack and KVM/QEMU, with the vRouter running in the host Linux OS and containing VRFs with virtual network forwarding tables. All containers in a pod share a networking stack with a single IP address (IP-1, IP-2 in the diagram), but listen on different TCP or UDP ports, and the interface of each networking stack is connected to a VRF at the vRouter. A process called kube-network-manager listens for network-related messages using the Kubernetes API and sends these into the Contrail Networking API. When a pod is created on a server, there is communication between the local kubelet and the vRouter agent via the Container Network Interface (CNI) to connect the new
interfaces into the correct VRFs. Each pod in a service is allocated a unique IP address within a virtual network, and also a floating IP address which is the same for all the pods in a service. The service address is used to send traffic into the service from pods in other services, or from external clients or servers. When traffic is sent from a pod to a service IP, the vRouter attached to that pod performs ECMP load balancing using the routes to the service IP address that resolve to the interfaces of the individual pods that form the destination service. When traffic is sent to a service IP from outside the Kubernetes cluster, the load balancing is performed by a gateway router that is peered with the Contrail Networking controller. Kubernetes proxy load balancing is not needed when Contrail Networking virtual networking is used in a Kubernetes cluster.

When services and pods are created or deleted in Kubernetes, the kube-network-manager process detects corresponding events in the Kubernetes API, and it uses the Contrail Networking API to apply network policy according to the network mode that has been configured for the Kubernetes cluster. The various options are summarized in the following table.

<table>
<thead>
<tr>
<th>Networking Mode</th>
<th>Network Policy</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kubernetes default</td>
<td>Any-to-any, no tenant isolation</td>
<td>Any container can talk to any other container or service</td>
</tr>
<tr>
<td>Namespace isolation</td>
<td>Kubernetes namespaces map to projects in Contrail Networking</td>
<td>Containers within a namespace can communicate with each other</td>
</tr>
<tr>
<td>Service isolation</td>
<td>Each pod is in its own virtual network and security policy is applied so that only the service IP address is accessible from outside the pod</td>
<td>Communication within a pod is enabled, but only the service IP address is accessible from outside a pod</td>
</tr>
<tr>
<td>Container isolation</td>
<td>Zero-trust between containers in the same pod.</td>
<td>Only specifically allowed communications between containers are enabled, even within a pod. Only specific pod to specific services may be enabled.</td>
</tr>
</tbody>
</table>

Contrail Networking brings many powerful networking features to the Kubernetes world, in the same way that it does for OpenStack, including:

- IP address management
- DHCP
- DNS
Contrail Networking and VMware vCenter

This section describes a use case where the Contrail vRouter is installed on ESXi hosts to provide virtual networking and service for virtual machines. Contrail Networking can also be used to configure physical switches to provide virtual networking between virtual machines and this is described in Fabric Management in Contrail Networking.

VMware vCenter is in widespread use as a virtualization platform, but requires manual configuration of a network gateway in order to achieve networking between virtual machines that are in different subnets, and with destinations external to a vCenter cluster. Contrail Networking can be deployed in an existing vCenter environment to provide all the networking features that were listed previously, while preserving the workflows that users may have come to rely on to create and manage virtual machines using the vCenter GUI and API. Additionally, support has been implemented for Contrail Networking in vRealize Orchestrator and vRealize Automation so that common tasks in Contrail Networking such as creation of virtual networks and network policies can be included in workflows implemented in those tools.

The architecture for Contrail Networking working with VMware vCenter is shown in Figure 21, below.
Virtual networks and policies are created in Contrail Networking, either directly, or using Contrail Networking tasks in vRO/vRA workflows.

When a VM is created by vCenter, using its GUI or via vRO/vRA, the vCenter plugin for Contrail Networking will see a corresponding message on the vCenter message bus, and this is the trigger for Contrail Networking to configure the vRouter on the server that the VM will be created on. Each interface of each VM is connected to a port group that corresponds to the virtual network that the interface is in. The port group has a VLAN associated with it that is set by the Contrail Networking controller using the "VLAN override" option in vCenter, and all the VLANs for the port groups are sent through a trunked port group into the vRouter. The Contrail Networking controller maps between the VLAN of an interface to the VRF of the virtual network that contains that subnet. The VLAN tag is stripped, and route look up in the VRF is performed as described in the section *Detailed Packet Processing Logic In a vRouter*.

Using Contrail Networking with vCenter gives users access to the full range of network and security services that Contrail Networking offers as described in earlier in this document, including zero-trust microsegmentation, DHCP proxy, DNS, and DHCP which avoids network flooding, easy service chaining, almost unlimited scale, and seamless interconnection with physical networks.

*Figure 21: vRouter runs in a virtual machine in vCenter environments*
Nested Kubernetes with OpenStack or vCenter

In the previous section, it was assumed that the KVM hosts where the containers run had been provisioned beforehand by some means. An alternative is to use OpenStack or vCenter to provision VMs in which the containers run, and for Contrail Networking to manage virtual networking between VMs created by OpenStack or vCenter and containers created by Kubernetes. This is illustrated in Figure 22, below.

The orchestrator (OpenStack or vCenter), Kubernetes Master, and Contrail Networking are running in a set of servers or VMs. The orchestrator is configured to manage the compute cluster with Contrail Networking, so there are vRouters on each server. VMs can be spun up and configured to run Kubelet and the CNI plugin for Contrail Networking. These VMs become available for the Kubernetes master to run containers in, with networking managed by Contrail Networking. Since the same Contrail Networking is managing the networks for both the orchestrator and Kubernetes, seamless networking is possible between VMs, between containers, and between VMs and containers.

In the nested configuration, Contrail Networking delivers the same levels of isolation as described previously, and it is possible for multiple Kubernetes masters to co-exist and for multiple VMs running Kubelet to run on the same host. This allows multitenant Kubernetes to be offered as a service.
Connecting to Physical Networks

In any data center, there is a need for some VMs to access external public IP addresses, and for users outside the data center to access some VMs via public IP addresses. Contrail Networking provides several ways to achieve this:

- VPN connection to a BGP-enabled gateway
- Source NAT in the vRouter
- Local gateway in the vRouter to the underlay fabric

Each of these is applicable in different use cases, and each has varying dependencies on configuration of external devices and networks.

The methods of connection to external networks are described in the following sections.

**BGP-Enabled Gateway**

One way of achieving external connectivity is to create a virtual network using a range of public IP addresses, and to extend the network to a gateway router. When the gateway router is a Juniper MX Series router, the configuration on the device can be done automatically by Contrail Networking. This is illustrated in Figure 23, below.

![Figure 23: Using BGP peering with floating IP addresses to connect to external networks](image-url)
Network A is defined in Contrail Networking, and contains a subnet of publicly addressable IP addresses. This public virtual network is configured in Contrail Networking to extend to the gateway router, which results in Contrail Networking automatically creating a VRF on the gateway with route target matching that of the virtual network (e.g. VRF labeled A). Contrail Networking configures this VRF with a default route that causes the route lookup for traffic arriving in the VRF from a vRouter on a compute node to occur in the main inet.0 routing table (which will contain routes to public destinations in the Internet). A forwarding filter is installed, which causes traffic arriving at the gateway with destinations in Network A to be looked up in the VRF that Contrail Networking created. The router advertises a default route via the VRF to the Contrail Networking controller.

Network A is configured to be a floating IP address pool in Contrail Networking, and when such an address is assigned to an existing VM interface, an additional VRF (e.g. for Network A) is created in the vRouter for the VM, and the interface is connected to the new, public VRF, in addition to being connected to the original VRF (green or red in Figure 23). VRFs for floating IP addresses perform 1:1 NAT between the floating IP address and the IP address configured on the VM. The VM is unaware of this additional connection and continues to send and receive traffic using the address for its original virtual network that it received via DHCP. The vRouter advertises a route to the floating IP address to the controller, and this route is sent to the gateway via BGP and it is installed in the public VRF (e.g. VRF A). The Contrail Networking controller sends the vRouter a default route via the VRF on the physical router and this is installed in the vRouter's public VRF.

The result of these actions is that the public VRFs on vRouters contain a route to a floating IP address via a local interface of a VM, and a default route via a VRF on the router. The VRFs on the gateway have a default route (implemented using filter-based forwarding) via the inet.0 route table, and have host routes to each allocated floating IP address. The inet.0 route table has routes to each floating IP network via the corresponding VRF.

Multiple separate public subnets can be used as separate floating IP address pools with their own VRFs when tenants own their own public IP address ranges (as shown in the diagram), and conversely, one floating IP address pool can be shared among multiple tenants (not shown).

In cases where a non-Juniper device is used, or Contrail Networking is not permitted to make configuration changes on the gateway, a BGP session, public network prefix and static routes can be set up on the gateway manually, or by a configuration tool. This method is used when the router is combining a provider edge (PE) router role for enterprise VPNs with a data center gateway role. Generally, in this case, the VRFs will created by a VPN management system. A virtual network in the Contrail Networking cluster will be connected into an enterprise VPN when a matching route target is configured in the virtual network, and routes are exchanged between the controller and the gateway/PE.
Source NAT

Contrail Networking enables networks to be connected via a source-based NAT service which allows multiple VMs or containers to share the same external IP address. Source NAT is implemented as a distributed service in each vRouter. The next hop for traffic being sent from a VM to the Internet will be the SNAT service and it will forward to the gateway of the underlay network with source address modified to that of the vRouter host and source port specific to the sending VM. The vRouter uses the destination port in returning packets to map back to the originating VM.

Routing in Underlay

Contrail Networking allows networks to be created that use the underlay for connectivity. In the case that the underlay is a routed IP fabric, the Contrail Networking controller is configured to exchange routes with the underlay switches. This allows virtual workloads to connect to any destination reachable from the underlay network and provides a much simpler way than a physical gateway to connect virtual workloads to external networks. Care must be taken that overlapping IP address are not connected into the fabric, so this feature is more useful for enterprises connecting cloud to legacy resources rather than multitenant service providers.

Fabric Management in Contrail Networking

A key feature in Contrail Networking is the ability to manage the full lifecycle of switches and routers from zero-touch provisioning through fabric configuration. Device lifecycle management features are based on open standards such as Ethernet VPN (EVPN), VXLAN, and NETCONF. The entire lifecycle of devices is supported: device discovery, provisioning base configuration, configuring underlay connectivity, and overlay networking for attached workloads in bare metal servers. This section describes the interaction between Contrail Networking and switches in an IP fabric as they are brought from factory default configuration to become a fully functioning environment.

Contrail Networking supports two modes of bringing a fabric under management:

- **Greenfield** – where devices begin with the factory default configuration
- **Brownfield** – where devices have management IP addressing and loopback interconnectivity already configured

Scope of Fabric Management

The scope of fabric management Contrail is summarized in Figure 24, below.
A fabric is a group of connected devices within which sets of devices perform different roles (gateway, spine, leaf). If the spine devices support gateway functionality (e.g. QFX10000 Series switches) the separate gateway layer may be omitted. Each device is assigned one or more routing/bridging roles, such as route reflector, SDN gateway, etc. Routing/bridging roles are described in detail in the following section. Contrail Networking can manage multiple fabrics and the connections between them. Additionally, it can manage connections to servers using VLANs or access ports, and use VXLAN virtual networking to connect groups of servers together, and can provide connectivity to external networks. Server management is described in detail in *Lifecycle Management and Virtual Networking for Bare Metal Servers*, below. Additionally, Contrail Networking can integrate with VMware vCenter and provide connectivity in the fabric for port groups created in vCenter. This is described in *Contrail Networking and VMware vCenter*.

The focus of this section is how a fabric is configured to be ready to support overlay networking between servers.

Figure 25 shows how Contrail Networking sets up a fabric to support overlay networking.
Each spine is connected to each leaf, and to each gateway, when separate gateways are in use. There may be multiple physical connections between devices; if so, Contrail Networking can configure these as link aggregation groups (LAGs). A logical interface is configured on each connection. Connected interfaces are assigned addresses from /31 subnets and a different subnet is used for each pair or set of connected interfaces. Each device is assigned a different autonomous system (AS) number, and an EBGP session using these runs over each connection to allow each loopback address to be advertised to all switches in the fabric. Connectivity between the loopback interfaces forms the underlay network; an IBGP mesh is used to distribute overlay routes for physical servers when they are attached to the fabric (described later in this document). Contrail uses route reflectors in the spine or gateway layer to distribute these overlay routes.

More information on using Contrail Networking to manage a fabric, see the Data Center: Contrail Enterprise Multicloud for Fabric Management solution guide. General information on fabric design, configuration, and operations may be found in the Cloud Data Center Architecture Guide and the book Data Center Deployment with EVPN/VXLAN, which are available on the Juniper website. Fabric management in Contrail Networking follows the design principles and configuration details laid out in these documents.
Key Concepts Used in Fabric Lifecycle Management

Before describing the fabric creation process in detail, some key concepts in Contrail Networking—roles, namespaces, and virtual port groups—must be introduced.

Roles

The fabric management feature of Contrail Networking uses a concept called “node profiles” to specify what roles and capabilities each device type can have. Each specific device model supported in Contrail Networking has a corresponding node profile which contains a list of the roles the device can perform. The roles have two parts: one that describes the location within the fabric (gateway, spine or leaf) and another that describes the network functions the device performs when in that location. For instance, a Juniper Networks QFX5100e-48s-6q switch can act as either a leaf or a spine and can provide access ports for servers as a leaf. In contrast, a Juniper Networks QFX5110-48s-4c switch can fulfill the same roles as the QFX5100e-48s-6q and can also perform centralized routing between VXLAN networks and act as a data center gateway.

The following section provides examples of entries in the Ansible configuration file that specify the roles available for different device types. In this file, “CRB” stands for centrally-routed bridging.

```
juniper-qfx5100e-48s-6q:
    - CRB-Access@leaf
    - null@spine
juniper-qfx5110-48s-4c:
    - CRB-Access@leaf
    - null@spine
    - CRB-Gateway@spine
    - DC-Gateway@spine
```

Further down the same configuration file, features configured for each role are specified. For instance:

```
null@spine:
    - basic
    - ip_clos
    - overlay_bgp
    - overlay_networking
CRB-Gateway@spine:
    - basic
    - ip_clos
    - overlay_bgp
    - overlay_evpn
    - overlay_evpn_gateway
    - overlay_security_group
```
Key Concepts Used in Fabric Lifecycle Management

- overlay_lag
- overlay_multi_homing
- overlay_networking
- overlay_evpn_type5

Each of these features has a corresponding Ansible playbook (with Jinja2 templates) which runs when the feature is present in a role that is applied to a device. The following table describes the various networking roles that are defined in Contrail Networking, and which physical roles they can apply to. Support for each networking role depends on the specific device model that is employed in a given physical role.

<table>
<thead>
<tr>
<th>Role</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>null</td>
<td>Applies only to spines when edge routing and bridging is used.</td>
</tr>
<tr>
<td>Route-Reflector</td>
<td>Specify at least one device in each fabric to act as a route reflector. Usually all spines or gateways are given this role.</td>
</tr>
<tr>
<td>CRB-Gateway</td>
<td>Centrally-routed bridging. Creation of logical routers in Contrail to connect virtual networks will result in VRFs containing IRBs for each network being created and subnet route distribution using Type 5 routes.</td>
</tr>
<tr>
<td>CRB-Access</td>
<td>Apply to leaf devices that will have bare metal servers attached to them.</td>
</tr>
<tr>
<td>CRB-MCAST-Gateway</td>
<td>Provides multicast protocol and ingress replication support.</td>
</tr>
<tr>
<td>ERB-UCAST-Gateway</td>
<td>Edge-routed bridging. IGMP snooping on access interfaces and logical routing support with Type 5 routes.</td>
</tr>
<tr>
<td>DC-Gateway</td>
<td>Devices that provide connectivity to external networks. Apply to devices in spine layer when a collapsed gateway architecture is used, or to separate gateway devices if they are present.</td>
</tr>
<tr>
<td>DCI-Gateway</td>
<td>Used for connectivity between fabrics.</td>
</tr>
<tr>
<td>AR-Replicator</td>
<td>Device performs assisted replication (AR) for BUM traffic.</td>
</tr>
<tr>
<td>AR-Client</td>
<td>Device sends BUM traffic to another device which performs AR.</td>
</tr>
</tbody>
</table>

NOTE: Normally either CRB roles or ERB roles are applied in leaf and spine switches, for all routing between virtual networks in a fabric, but this is not mandatory, and both types of roles can coexist in the same fabric.

Namespaces
Namespaces are pools from which values can be drawn and allocated by Contrail Networking. These values are used to specify, for instance, the subnet from which loopback addresses should be allocated, or a number range from which BGP autonomous system (AS) numbers should be allocated.
for the point-to-point connections between spine and leaf devices. Namespaces are generally specified when a fabric is first created.

The process for creating a greenfield fabric is described below, with annotation for the differences that relate to the brownfield scenario.

**Virtual Port Groups**

Virtual Port Groups (VPG) are groups of switch ports which will be configured to form an EVPN Segment Identifier (ESI) Link Aggregation Group (LAG). Interfaces with the same ESI are generally on the same server, and Link Aggregation Control Protocol (LACP) is run on these interfaces so that the server can support bond interfaces with load balancing across the links and failover in the case that a link goes down.

Each VPG has one or more VLANs (including untagged) associated with it and each is associated with a Contrail Networking virtual network, a set of security groups, and a port profile (which currently just contains storm control settings). The VLANs in the VPG should match the VLANs that are configured on the server ports. The virtual network subnets of the VPG must match those configured on the server if Contrail Networking is used to provide Layer 3 services such as logical routing between virtual networks managed by Contrail Networking, or for configuring a gateway router in order that the workloads attached to the fabric can access external networks.

**Procedure for Fabric Creation**

The fabric creation and management process includes four main stages:

- **Fabric creation:** Where the namespaces (allocation pools for IP addresses, etc.) are specified.
- **Device discovery:** Interfaces and connectivity are detected; underlay connectivity and overlay control plane are specified.
- **Role Assignment:** The user specifies fabric role and routing/bridging roles for each device.
- **Autoconfiguration:** Ansible playbooks are run to configure the underlay connectivity, overlay control plane, and the roles on each device.

The details of each stage are described in the following sections.

**Fabric Creation**

In the first stage of fabric configuration, the Contra Command interface is used to configure the following information:

- Name for the fabric
- Types of devices (node profiles)
Procedure for Fabric Creation

- Management subnet with gateway
- Underlay AS range
- Fabric subnet(s) for spine-leaf point-to-point connections
- Loopback subnet(s)

Once this information has been entered, the user can proceed to the device discovery stage.

Device Discovery

In a greenfield deployment, devices are racked with the factory default configuration, which periodically issues DHCP requests from the management interface. During racking, management interfaces should be connected to a VLAN that provides access to the Contrail Networking controller. Embedded in the cluster are parts of the bare metal server management function of OpenStack, including a DHCP server. When the management subnet is specified as a namespace, Contrail Networking configures the subnet and gateway into the configuration file of the DHCP server. When the racked devices next issue a DHCP request, the DHCP server responds with an IP address and the default gateway.

During the discovery phase, Contrail Networking detects when a device has been sent its management IP address and can run an Ansible playbook that pushes some basic configuration to the device, including the management IP address, and enabling NETCONF, SNMP, and LLDP. A subsequent playbook retrieves facts about the device, including its name, the model, and a list of its interfaces. Neighbor connectivity is retrieved from the LLDP tables using SNMP. The user can enter the number of devices in the fabric so the discovery process will end once that number of devices has been found.

The next stage, configuring underlay connectivity, is done by configuring point-to-point connections between spine and leaf devices, configuring loopbacks in each device, and configuring an EBGP session between connected devices. This causes each device to receive routes to all other devices in the fabric using neighbors as next hops when there isn’t a direct connection. This completes the underlay configuration.

In a brownfield scenario, devices are already configured with management connectivity. A ping sweep discovers the devices, followed by the interface discovery process described above.

Role Assignment

Once device models appear in the Contrail Command GUI following the discovery phase, it becomes possible to assign roles to each device. As described previously, the roles have two parts: the physical role within the fabric (spine/leaf) and the routing/bridging role. After specifying the physical role of a device, the user is presented with the routing/bridging roles available for that device model with the assigned physical role.
Autoconfiguration
Once roles are specified for all devices, the user presses the Autoconfigure button. A set of playbooks is then run to configure the specified role on each device.

First, the point-to-point connections between neighboring devices are configured, together with EBGP sessions over those connections to enable neighbor-to-neighbor connectivity. The overlay control plane is then created by configuring IBGP sessions between device loopbacks in the fabric using the AS number specified during fabric creation. A sequence of Ansible playbooks is run to achieve this.

At this point, the environment is ready for bare metal servers (BMS) or VMware ESXi servers to be attached to switch ports and for them to be placed in VXLAN virtual networks by configuring interfaces on those ports.

Device Operations
Contrail Networking supports the following device operations:

- Addition/deletion of devices in a fabric
- Software upgrade to new OS version for devices
- RMA procedure for replacement of a device within a fabric

Lifecycle Management and Virtual Networking for Bare Metal Servers
Contrail Networking provides lifecycle management for bare metal servers, including provisioning the OS and configuring the ports of the switches they are attached to. The same networking and security features that exist in Contrail Networking for virtual machines and containers are implemented in the interfaces of switches to apply to bare metal servers connected to them. Contrail Networking uses EVPN sessions with fabric switches to enable seamless networking between physical and virtual workloads.

The functional architecture for managing, provisioning, and networking of bare metal servers is shown in Figure 26.
Bare metal server provisioning is handled by OpenStack components embedded in Contrail Networking. Contrail Command provides a GUI for the features of Glance, Nova, and Ironic that are used by Contrail Networking. Ironic is the service in OpenStack that allows bare metal servers to be managed using the Nova service. When Nova receives an instruction that concerns a bare metal server, Ironic takes over the fulfillment. The Contrail Service Node (CSN) is a DHCP proxy for bare metal servers to receive IP addresses from Contrail's IP address management. The sequence of events for attaching a server to a fabric and configuring its networking has two main steps:

- Add the server to the infrastructure inventory by identifying which switch ports it is attached to, the MAC address for each interface, and whether bonded and/or multi-homed connections are used.
- Identify an image to be provisioned and a server on which to provision it. Configure the switch interface with VXLAN networking and provision the server from a Glance image.

Note that the physical server and the operating system running on it are treated as separate objects in Contrail Networking.

Servers can be fully managed by Contrail Networking, which can provision the operating system in addition to providing connectivity using VXLAN overlay networks that are configured in the switches to which servers are connected. Existing servers with already-configured IP addresses can also be connected into Contrail virtual networks.
The sequence of operations involved when Contrail Networking manages lifecycle management and virtual networking for physical servers is described in detail in the white paper Fabric and Server Lifecycle Management with Contrail Networking which is available on the Juniper website. The following sections summarize some of the content in the white paper concerning packet flows in virtual networking for physical servers.

**Virtual Networking for Physical Servers**

This section describes packet flow between servers on the same and different virtual networks, and between physical servers and virtual workloads (virtual machines or containers).

**Packets Between Servers in the Same Virtual Network**

Two servers attached to switches in a fabric and configured to be in the same virtual network are shown in Figure 27. The servers have been provisioned by Contrail Networking, which has also configured the VTEPs on each switch. For the purposes of this document, the route target of the virtual network will be called "Red," although its actual value will be in the form of target:xxx:yyy, where xxx and yyy are numeric values, and the VNI will also be called "Red." As a convention in this document, a letter in parenthesis, e.g. (R), indicates the route target or VNI attached to a route update.

![Figure 27: Connectivity between two servers using a VXLAN overlay tunnel](image)

There are IBGP sessions between switches (typically implemented using route reflectors in spine switches), and the various routes to servers attached to the switches are exchanged between them (note that leaf-to-leaf connections always physically traverse a spine switch). As explained in detail in the white paper referenced above, routes to servers are installed and advertised when traffic is sent into the network and the bridging table in a switch gets populated. In this example, each switch
advertises a route to its connected server via a Red VXLAN tunnel with itself as the tunnel destination. When a packet destined for server S2 is sent from S1, the leaf switch L1 finds a route to S2 in its routing table via a VXLAN tunnel to switch L2, and there will be an ECMP route to S2 via each of the spine switches. The leaf selects a route to a spine, and then forwards the packet to L2 inside VXLAN encapsulation with the VNI set to Red to that spine, which then routes it to S2. S2 decapsulates the packet and sends it into the server interface.

Since ECMP is used in both directions, forward and reverse traffic can pass through different spine switches.

**Packets Between Servers in Different Networks**

If communication between servers in different networks is required, routing must be configured between them. This is done in Contrail Networking by configuring a logical router that includes the desired networks. When a logical router is created, Contrail uses NETCONF to configure switches in the fabric with a VRF containing IRBs for each of the virtual networks that are to be connected. Each IRB is configured with the default gateway address for its network, and is also configured in a VTEP with VNI for that network. These VRFs can be specified to be created in spine switches (centrally-routed bridging - CRB), or in leaf switches (edge-routed bridging - ERB).

The CRB use case is shown in Figure 28, below.

![Figure 28: Logical router is implemented as VRFs with IRBs in spine switches for CRB](image-url)
When the IRBs are configured in each VRF, BGP routes for the IRB gateway address are sent by the spine to each of the leaf switches. Leaf switches select which spine to use via ECMP. This means that the forward and reverse traffic can pass through different spines.

When ERB is selected, and a logical router is created in Contrail Networking, the corresponding VRFs are configured in each leaf switch that has a server interface in a network that was configured in the logical router. This is shown in Figure 29, below.

Traffic sent by a server in one network with a destination in another network is routed locally in the local leaf switch and then sent in a VXLAN tunnel with the VNI of the destination network. The leaf-to-leaf traffic of the VXLAN tunnels is routed in the spine switches and since the leaf switches will use ECMP load balancing across spines, forward and reverse traffic can pass through different spines.

Traffic Between a Physical Server and a VM in the Same Virtual Network
Contrail Networking allows physical and virtual workloads to be interconnected seamlessly using overlay networking.

Figure 30 shows a virtual machine and a physical server with interfaces in the same network. The diagram shows routes being exchanged via the Contrail Controller, which mediates between the XMPP messages used by the vRouter and EVPN routes used on the switch.
Traffic between VM1 and S2 is carried in a VXLAN tunnel terminating in the vRouter on one side and the leaf switch on the other side.

**Traffic Between a Physical Server and a VM in Different Virtual Networks**

This section describes what happens when a physical server is connected to a switch that is running EVPN to manage VXLANs, and the server is connected to a VTEP with a different VNI (in this case, Green) than that of the VM with which it needs to communicate.

Figure 31 shows how a logical router is implemented as a VRF with IRBs, and how routes are exchanged when centrally routed bridging is used.
The spine switches are configured to use EVPN as the control plane, and with the IP addresses of control nodes in the Contrail Controller as peers. When the Green network is created in Contrail Networking, the administrator specifies that the network should be extended to the spine switches. This causes Contrail Networking to create a VRF routing instance on each spine switch. Each VRF contains two IRBs, which are configured with the gateway addresses of the Red and Green networks and with the Red and Green subnets. Routing towards the virtual environment is in the main routing table for the VRF, and outgoing traffic is configured to use VXLAN using a special VNI used for all virtual networks connected by the logical router. A different special VNI (in this case, Blue) is used for each logical router. On the vRouter running a Red VM, an additional VRF is created with that special VNI, and a default route is installed in the Red VRF such that traffic destined for the Green network is sent into the special VRF, then to the VRF in the spine switch, and finally on to the destination.

When edge-routing bridging is used, the logical router is placed on the leaf device together with the associated IRBs and VTEPs.

Physical Network Configuration for VMware vCenter

Contrail Networking Release 1910 and later supports integrating VMware vCenter with Contrail Networking fabric management. The integration enables Contrail Networking to configure switches to implement the connectivity expected for Distributed Port Groups (DPG) that VM interfaces are configured into. This is achieved using a plugin to vCenter, Contrail vCenter Fabric Manager (CVFM), which is deployed for this integration. This plugin monitors VM change events (create/delete/modify) and causes Contrail Networking to create corresponding Virtual Port Groups (VPG) when a VM with interfaces in a new DPG is started on an ESXi host. The VPG is configured with the VLAN specified in the DPG and uses a VXLAN virtual network to connect all the VMs that are in the same DPG. This procedure is completely non-intrusive to the VMware vCenter system and the ESXi hosts.

CVFM Design Overview

Figure 32 depicts the CVFM plugin installed on the Contrail Networking control node. The CVFM plugin detects changes in the vCenter environment and pushes the new configurations to the Contrail Device Manager. The Contrail Device Manager then pushes these configurations to fabric devices such as QFX Series switches.
The leaf and spine switches (QFX Series) are connected to virtual machines in the ESXi host environment. VLANs are configured on the DPG of these QFX Series switches. The CVFM plugin automatically adds and removes configurations of the VLANs when network change events are seen in vCenter.