Chapter 1

Foundation Principles

2019-08-18

# Containers overview

## Understanding containers

Two decades ago Virtualization was the most fashionable keyword in IT as there was a revolution in the way we build servers. It was mainly about the adoption of Virtual machine instead of dedicated physical server in building applications Back in that time Virtual Machine was a clear winner (also valid today) when it comes to scaling, portability, capacity management, cost, ..,etc and you can find tones of comparison between the two approaches

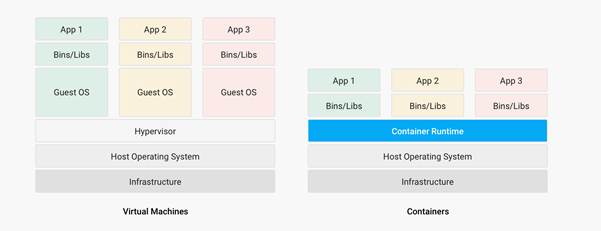
If Virtualization was the keyword that sums it up, the keywords of today are Cloud , SDN and Containers

Today, the heavily discussed comparison would be between VM and Containers promising a new way to build and scale applications. While many small organizations are thinking of containers as something too early to adopt, the simple fact that Google stated that “From Gmail to YouTube to Search, everything at Google runs in containers, we run two billion container a week” might give you a clue where the industry is heading

But what is Container and how is it comparable to VM?

From technical prospect container is rooted in the Namespaces and Cgroups concept in Linux but the name might be inspired by the Actual metal cargo shipping containers that you see on ships As both share the ideas of isolating contents , carrier independence , easy movement , …,etc

Container is a logical packaging mechanism, you can think of it as Lightweight virtualization that runs an application and its dependencies in the same operating system but in different context which remove the need to replicate an entire operating system as shown in the figure 1.1 By doing so Application would be confined in a lightweight package that could be developed/tested individually then implemented and scaled much faster than the tradition VM as Developer just need to build/configure this light piece of software. Currently most of the application has been containerized and publicly available. And finally there is no need to manage/support the application per OS



Container vs. VM

Many developers would call the Container Runtime in figure 1.1 “the Hypervisor of Containers” Although this is not a technically correct term, but it may be useful only to visualize the hierarchy.

As in the VM technologies, the most common Hypervisors are KVM and VMware ESX/ESXi. In the Container technologies Docker and Rkt are the most known with Docker being the most widely deployed one. Before we give an overview of Docker let’s see some useful numbers in comparing VM vs Container

## Juniper VSRX vs CSRX

Currently most of common applications such as Redis, Ngnix, Mongo , MySQL, WordPress, Jenkins , Kibana and Perl have been containerized and offered publicly in <https://hub.docker.com> allowing developers to build/test application quickly

There are a lot of tests that compare performance and scaling for a giving application when it runs on Container vs VM. All of this comparison stress on the benefits of running your application in container.

But what about network functions NFV such as firewall, NAT, Routing, …,etc? When it comes to VM based NFV, most of network vendors already implemented a Virtualized flavor of the hardware equipment that could be run on the hypervisor of standard x86 hardware. VSRX is a Juniper SRX Series Services Gateway in a virtualized form factor built on Junos, and delivers networking and security features similar to those available for SRX as for containerized based NFV, it’s the new trend. Juniper CSRX is the industry first containerized firewall offering a Compact footprint, high-density firewall for virtualized and cloud environments Table 1.1 showing a comparison between CSRX vs. VSRX which will you can see the idea of CSRX being lightweight NFV

CSRX vs. VSRX

|  |  |  |
| --- | --- | --- |
|  | vSRX | cSRX |
| Use Cases | Integrated routing, security, NAT, VPN, High Performance | L4-L7 Security, Low Footprint |
| Memory Requirement | 4GB Minimum | In MB’s |
| NAT | Yes | Yes |
| IPSec VPN | Yes | No |
| Boot-up Time | ~minutes | <1second |
| Image size | In GB’s | In MB’s |

**Note**

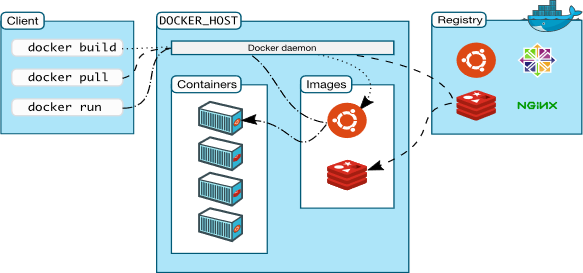
In micro services techinque the application would be split into smaller services with each part(container in that case) is doing specific job.

## Understanding Docker

As we have discussed, containers allow a developer to package up an application with all of the parts it needs, such as libraries and other dependencies, and ship it all out as one package Docker is the software to facilitate creating, deploying, and running containers

The starting point would be the source code of the Image “Docker file” from them you can build the “Docker Image” this image can be stored and distributed to any registry -most common Docker hub- and finally you use this image to run the containers.

Docker uses a client-server architecture. The Docker client and daemon can run on the same system, or you can connect a Docker client to a remote Docker daemon The Docker daemon, does the heavy lifting of building, running, and distributing your Docker containers. The Docker client and daemon communicate using a REST API, over UNIX sockets or a network interface.



Docker architecture

Containers doesn’t exists in a vacuum and in production you won’t have just one host with multiple container but multiples of hosts running 100s if not 1000s of containers, which raise two important sets of questions

1. How these containers communicate with each other on the same host or in different host as well with outside world? Basically, the networking parts of containers
2. Who determine which containers get launch over which host? Based on what? Upgrade ? Number of containers per application? Basically, who orchestrate that ?

we will try to answer these two questions in detail for the rest of the book but if you want a quick answer just think of Contrail and Kubernetes!!

Let’s start first lay the basics foundation of contrail

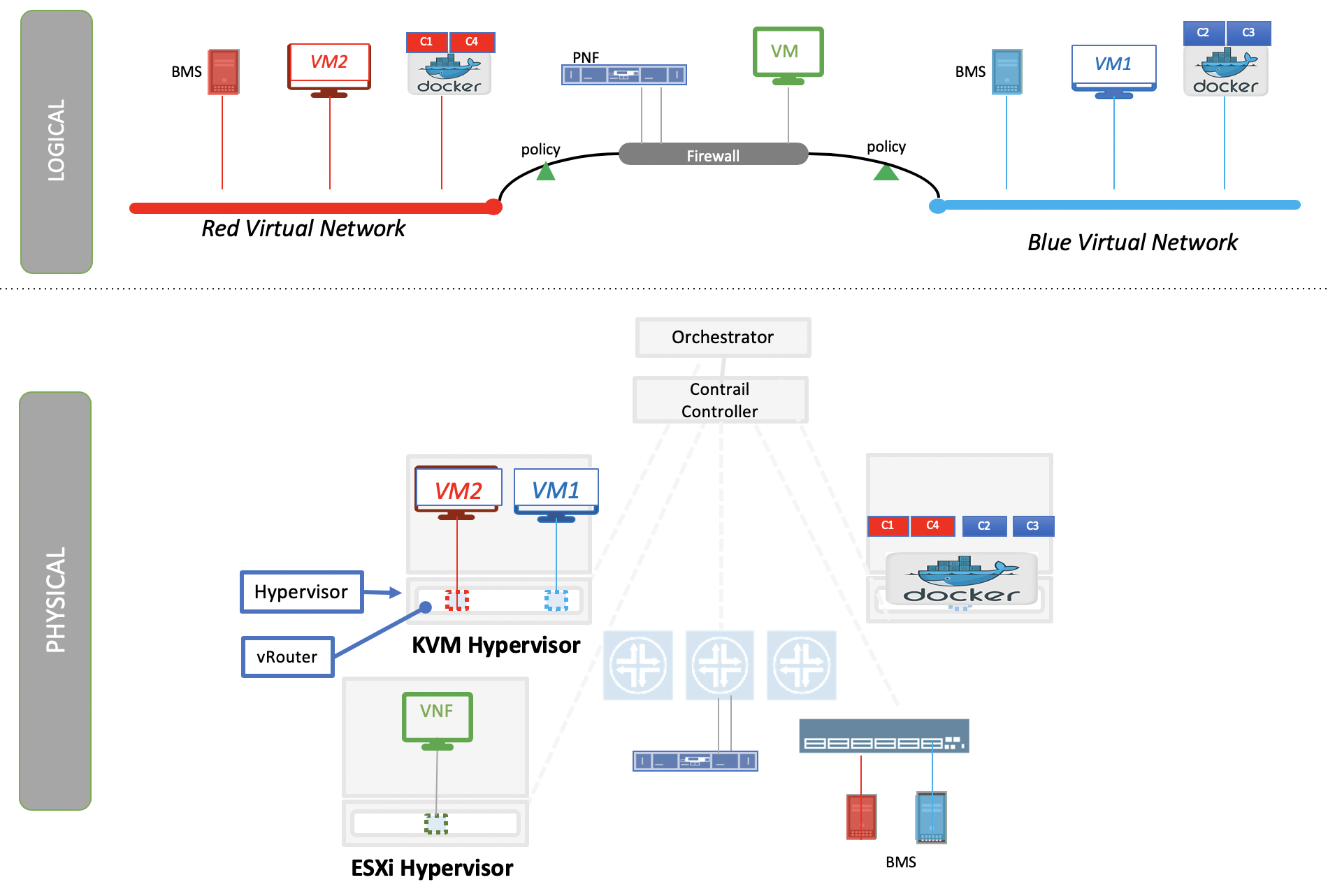
# Contrail overview

Contrail provides dynamic end-to-end networking, networking policy and control for any cloud, any workload, and any deployment, from a single user interface.

Although our focus in this book would be building a secure container network orchestrated by Kubernetes but contrail can build Virtual networks that integrate container, VM and BMS

Virtual Networks (VNs) are a key concept in the Contrail System. Virtual networks are logical constructs implemented on top of the physical networks. Virtual networks 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 virtual networks. Each virtual network is isolated from all the other virtual networks unless explicitly allowed by network policy. Virtual networks can be extended to physical networks using a gateway. Finally, Virtual networks are used to build service-chaining.

as shown in the figure 1.3, the Network operate only deal with the logical abstraction of the network then contrail do the heavy lifting which include but not limited to building polices, exchanging routes, building tunnels on the physical topology.



Virtual networks

## Contrail Architecture Fundamentals

Contrail run in Logically centralized, physically distributed model as it has two main components, Contrail controller and Contrail vrouter the Controller is the Control and management plane that Manages/configures the vrouter and Collects/presents analytics Contrail vrouter is the Forwarding plane that Provides Layer 2/3 services , Distributed firewall capabilities and Implements policies between virtual networks

Contrail integrates with many orchestrator such as OpenStack , VMware , Kubernetes , OpenShift and Mesos and use multiple protocols to provide SDN to these orchestrators as shown in figure 1.4 where

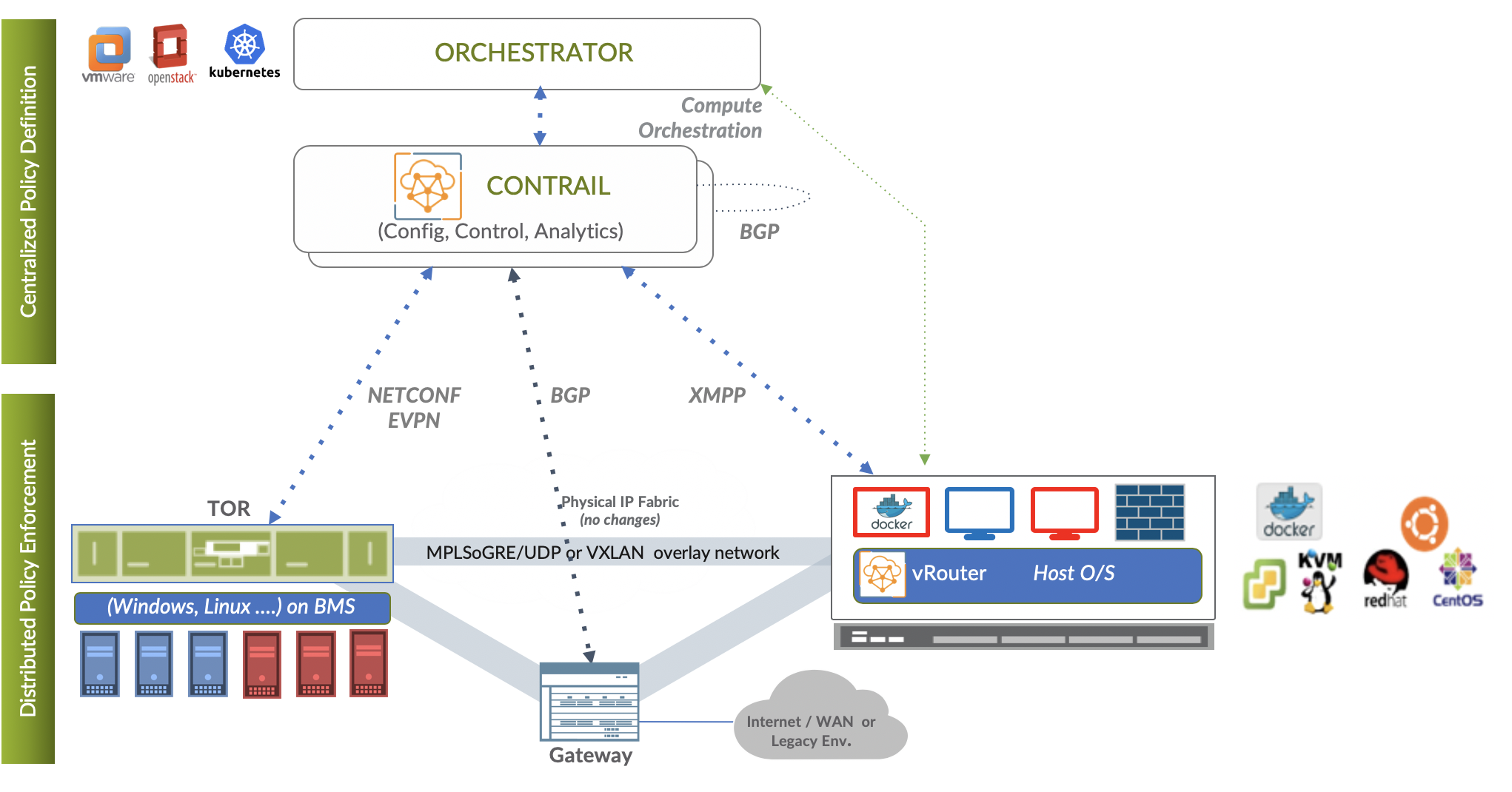
XMPP : Extensible Messaging and Presence Protocol (XMPP) is an open XML technology for real-time communication defined in RFC 6120, in Contrail it offers two main functionality, distributing routing information and pushing configuration, which are similar to what IBGP do in MPLS VPNs model plus NETCONF in device management.

BGP: is used to exchange route with physical router and in same case Contrail device manager can use Netconf to configure this Gateway

EVPN: Ethernet VPN is a standards-based technology RFC 7432 that provides virtual multipoint bridged connectivity between different Layer 2 domains over an IP network. Contrail controller exchange EVPN routes with TOR switches (acting as L2 VXLAN GW) to offer faster recovery with active-active VXLAN forwarding

MPLSoGRE/UDP or VXLAN: are three different kind of overlay tunnels to carry traffic over IP network. They are all IP-UDP packet but in VXLAN we use the VNI values in VXLAN header for segmentation where in MPLSoGRE and MPLSoUDP we use the MPLS label value for segmentation

To simplify the relation between contrail vrouter, contrail controller and the IP Fabric from the prospective of the Architecture prospective, let’s compare it to MPLS VPN model in any services provider vrouter is like PE router and the VM/container is like CE but vrouter is just a slave of contrail controller. and when it comes to BMS the TOR would be the PE



Contrail architecture

**Note**

we will be using the words “compute node” and “host” interchangeably in this book. Both would mean the entity will host the containers any container needs a compute node to host it. This host could be a physical server in your DC, or a VM (VM either in your DC or in public cloud).

## Contrail VRouter

Contrail vRouter is contrail components on compute node/host as shown in figure 1.3

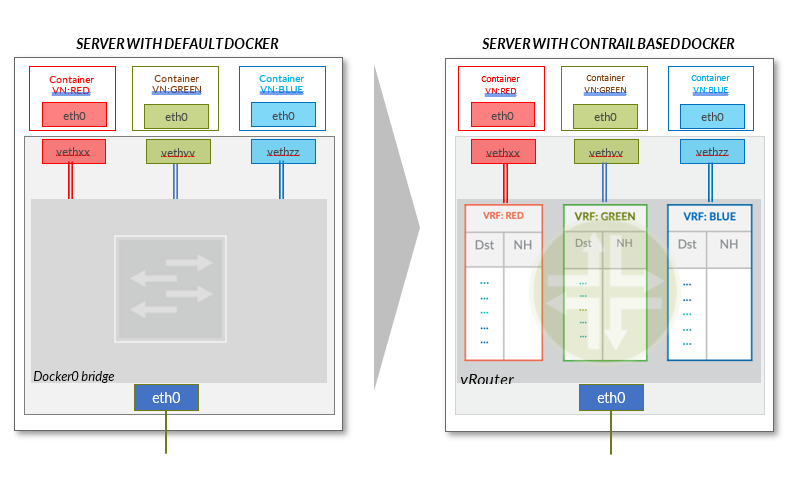
for a compute node in the default docker setup, containers on the same host communicate with each other as well with other containers/services hosted on other host with Docker bridge but with contrail networking, on each compute the vrouter creates VRF per virtual network offering long list of feature as will discuss

From the prospective of control plane the vrouter would

* Receive low-level configuration (routing instances and forwarding policy)
* Exchange routes Install forwarding state into the forwarding plane. Report
* analytics (logs, statistics, and events)

From the prospective of data plan the vrouter would

* Assign received packet from the overlay network to a routing instance based
* on the MPLS label or Virtual Network Identifier (VNI). Proxy DHCP, ARP, and
* DNS. Apply forwarding policy for the first packet of each new flow then
* program the action to the flow entry in the flow table of the forwarding
* plan. Forwarding the packer after a Destination address lookup (IP or MAC)
* in the Forwarding Information Base (FIB) Encapsulating/decapsulating packets
* sent to or received from the overlay network.



Docker and contrail vRouter