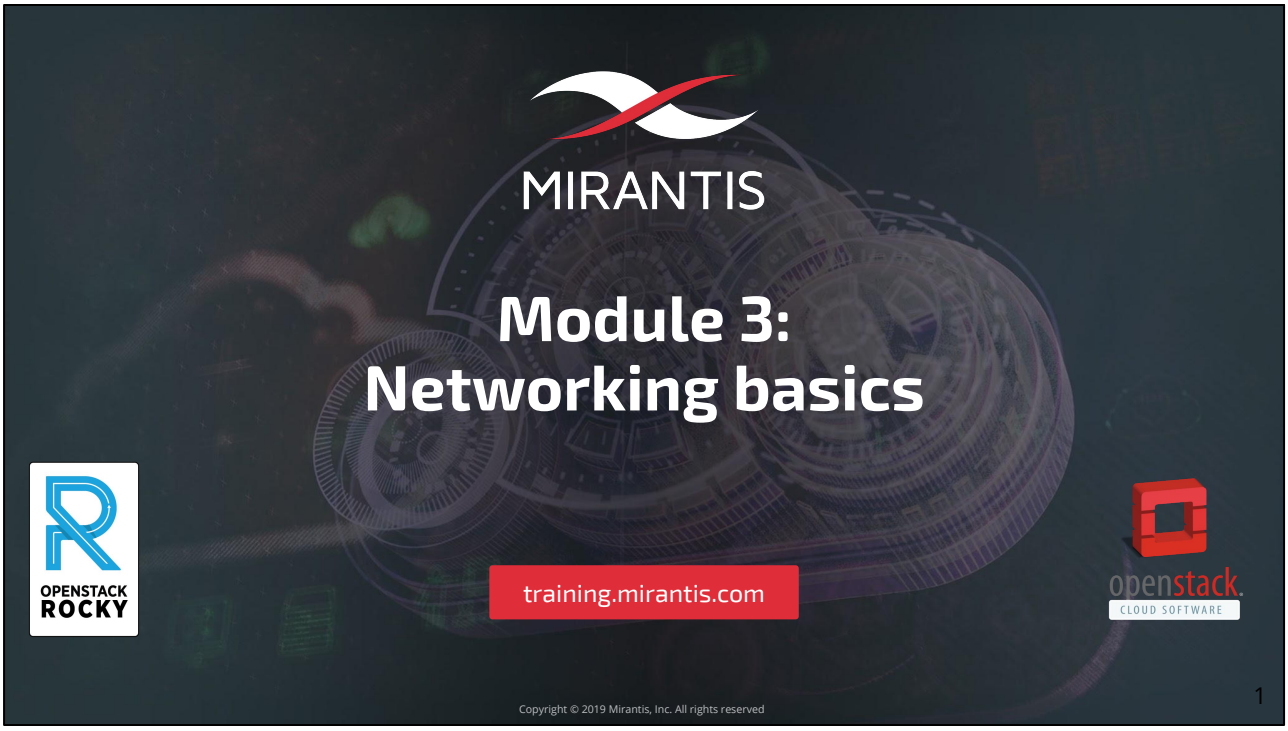


Objectives

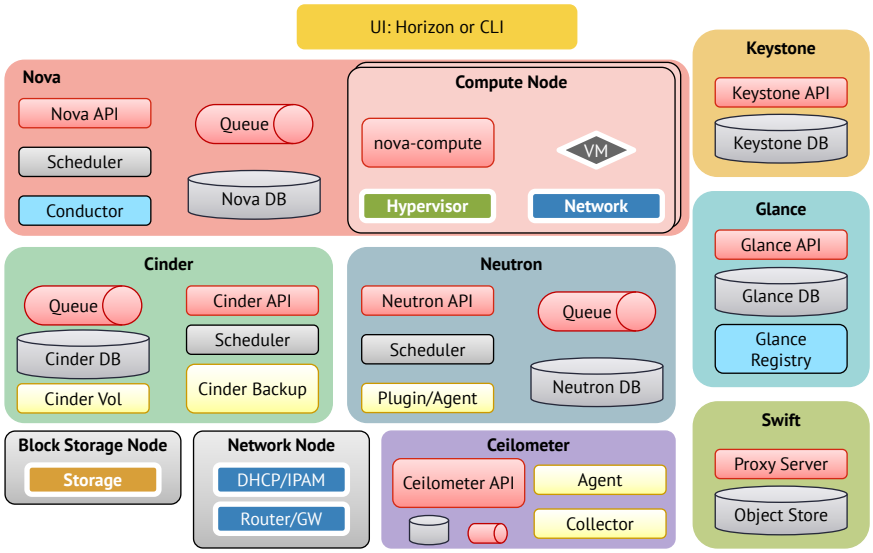
At the end of this presentation, you should be able to:

- Describe the 5 networks that *might be seen* in an OpenStack deployment
- Describe available options for switch virtualization
- Describe supported network protocols, including options using overlay networks (*tunneling*)

OpenStack networks

Networking in the large

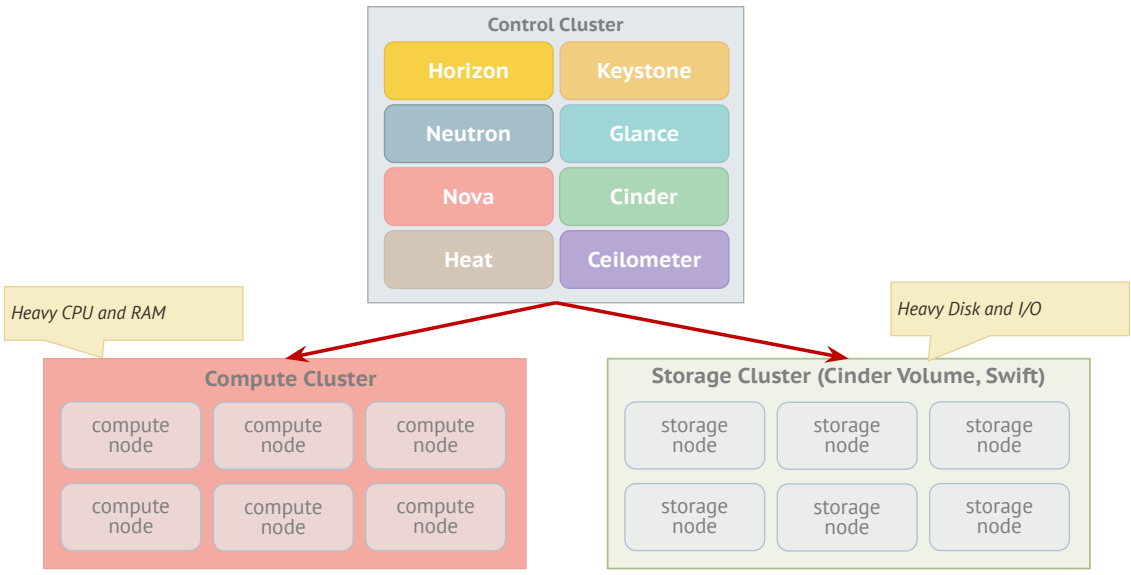
OpenStack components



OpenStack is the integration of many independent *programs*. This slide shows the most commonly used OpenStack components (Keystone, Glance, Nova, Neutron, Cinder, etc.)

Each program is comprised of several python daemons, plus a database, and, optionally, a message queue. The python daemons can run on the controller, compute, storage, or network nodes.

OpenStack: Logical deployment topology



MIRANTIS

5

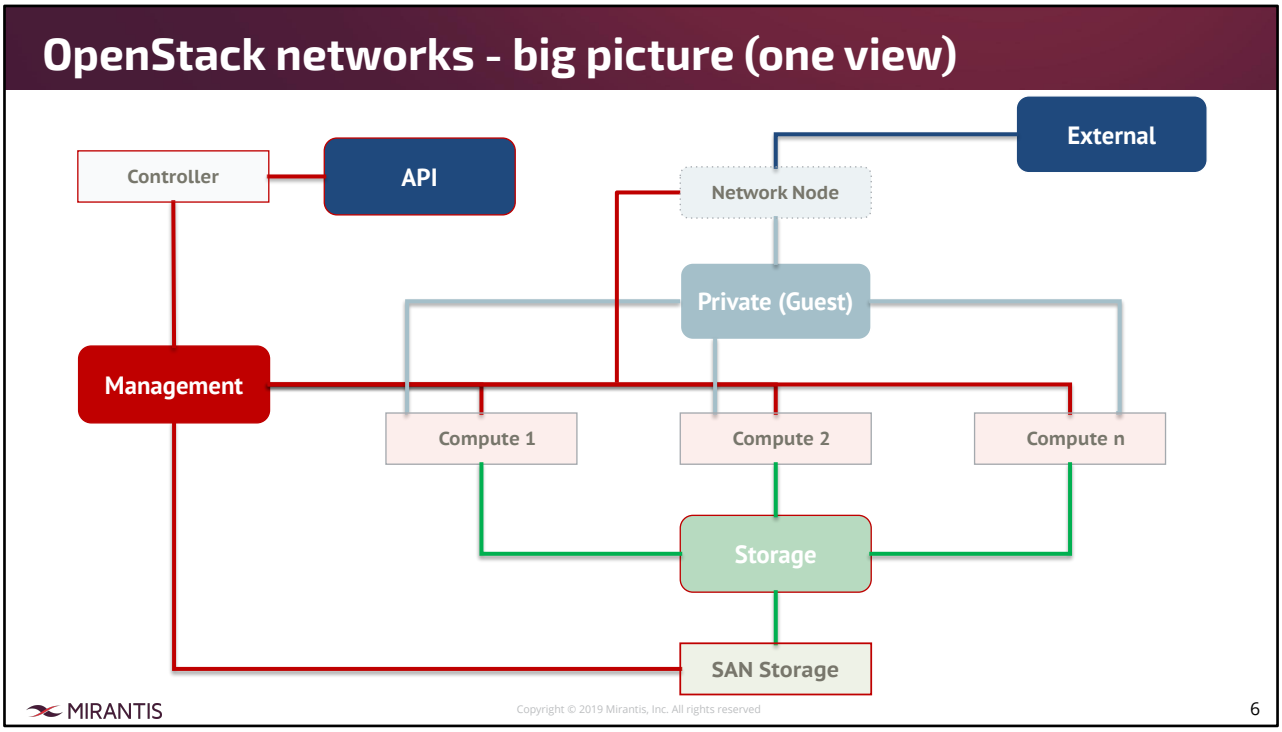
A typical deployment of OpenStack consists of at least 3 nodes (or clusters):

- Control node (cluster) is the brain of OpenStack. All components have processes running here.
- Compute node (cluster) interfaces with the hypervisor. You have a unique compute node per type of hypervisor.
- Storage node (cluster) handles all Cinder and Swift storage requests. You have a unique storage node per type of storage used (LVM, NetApp, Hitachi, and so on)

Each node (cluster) can be virtualized on a single server or VM for lab purposes. Production environments often segregate and customize services, *separating them due to hardware differences*.

The motivation for separation is:

1. The hardware and resource demand is different for each type of service. For example, lots of CPU and memory for compute nodes versus disk and I/O needs on storage nodes.
2. You don't want to affect core services by move, adds, changes on compute or storage resources
3. Generally a many:one relationship between compute and storage vs. shared / platform services



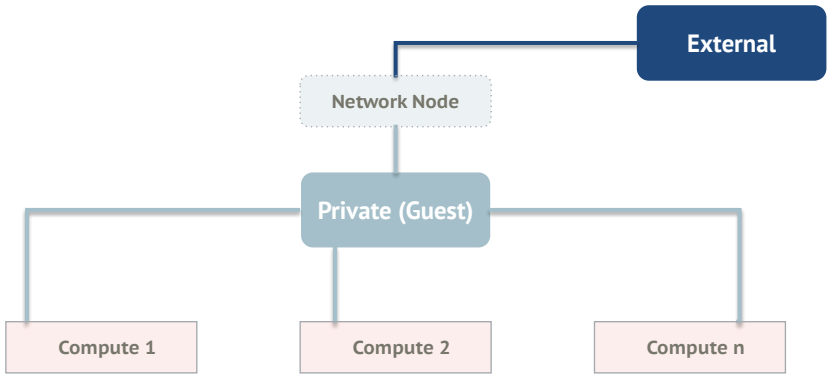
A standard OpenStack networking setup has several distinct physical data center networks:

- Management network
  - Used for internal communication between OpenStack Components. The IP addresses on this network should be reachable only within the data center and is considered the Management Security Domain.
  - Hosts traffic among the OpenStack components that manage the entire operation of the cloud infrastructure.
- Private (Guest) network
  - Hosts traffic among the virtual machines (VMs) in the OpenStack cloud. This network is used as internal network for traffic between virtual machines in OpenStack, and between the virtual machines and the network nodes that provide L3 routes out to the public network.
  - Used for VM data communication within the cloud deployment. The IP addressing requirements of this network depend on the OpenStack Networking plug-in in use and the network configuration choices of the virtual networks made by the tenant. This network is considered the Guest Security Domain.
- External (public) network
  - Used to provide VMs with Internet access in some deployment scenarios. The IP addresses on this network should be reachable by anyone on the Internet. This network is considered to be in the Public Security Domain.
  - This network is connected to the controller nodes (so users can access the OpenStack interfaces) and to the network nodes (to provide VMs with publicly routable traffic functionality).
- API network
  - Exposes all OpenStack APIs, including the OpenStack Networking API, to

    - tenants. The IP addresses on this network should be reachable by anyone on the Internet. This may be the same network as the external network, as it is possible to create a subnet for the external network that uses IP allocation ranges to use only less than the full range of IP addresses in an IP block. This network is considered the Public Security Domain.
- Storage network
    - Hosts traffic between the VMs and their application datasets that are on external storage systems.

In many cases, the Management and API networks are combined.

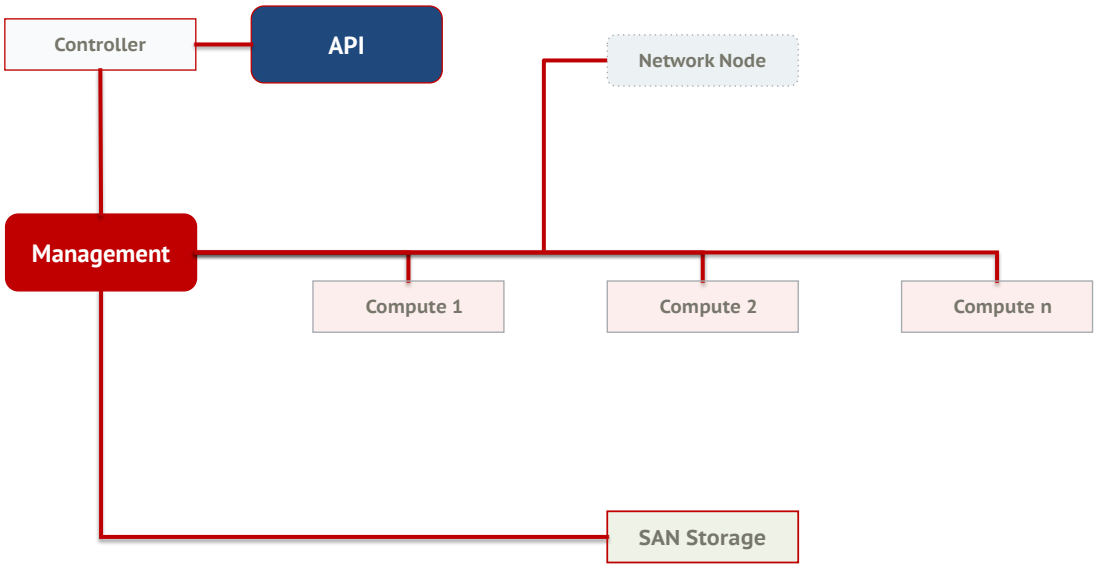
OpenStack networks - managed by Neutron



MIRANTIS

- Private (Guest) network
  - Hosts traffic among the virtual machines (VMs) in the OpenStack cloud. This network is used as internal network for traffic between virtual machines in OpenStack, and between the virtual machines and the network nodes that provide L3 routes out to the public network.
  - Used for VM data communication within the cloud deployment. The IP addressing requirements of this network depend on the OpenStack Networking plug-in in use and the network configuration choices of the virtual networks made by the tenant. This network is considered the Guest Security Domain.
- External (public) network
  - Used to provide VMs with Internet access in some deployment scenarios. The IP addresses on this network should be reachable by anyone on the Internet. This network is considered to be in the Public Security Domain.
  - This network is connected to the controller nodes (so users can access the OpenStack interfaces) and to the network nodes (to provide VMs with publicly routable traffic functionality).

OpenStack control plane - management network

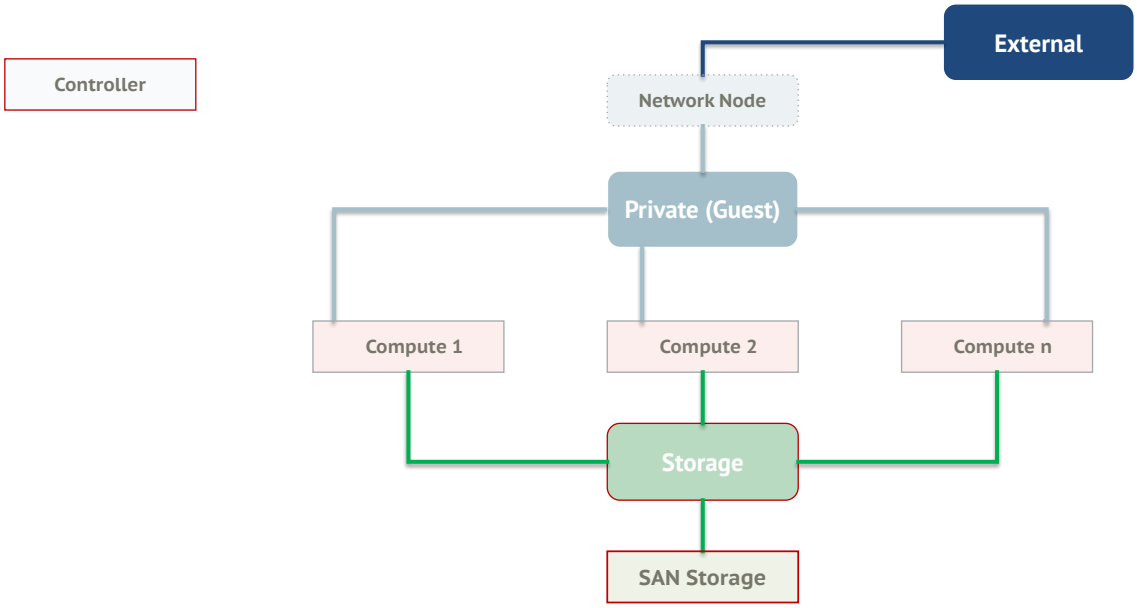


Management network

- Used for internal communication between OpenStack Components. The IP addresses on this network should be reachable only within the data center and is considered the Management Security Domain.
- Hosts traffic among the OpenStack components that manage the entire operation of the cloud infrastructure.

Many of the ***Control plane*** services (services, responsible for create, read, update and delete (CRUD) operations, metering, monitoring, and so on) use the Management network..

OpenStack data plane



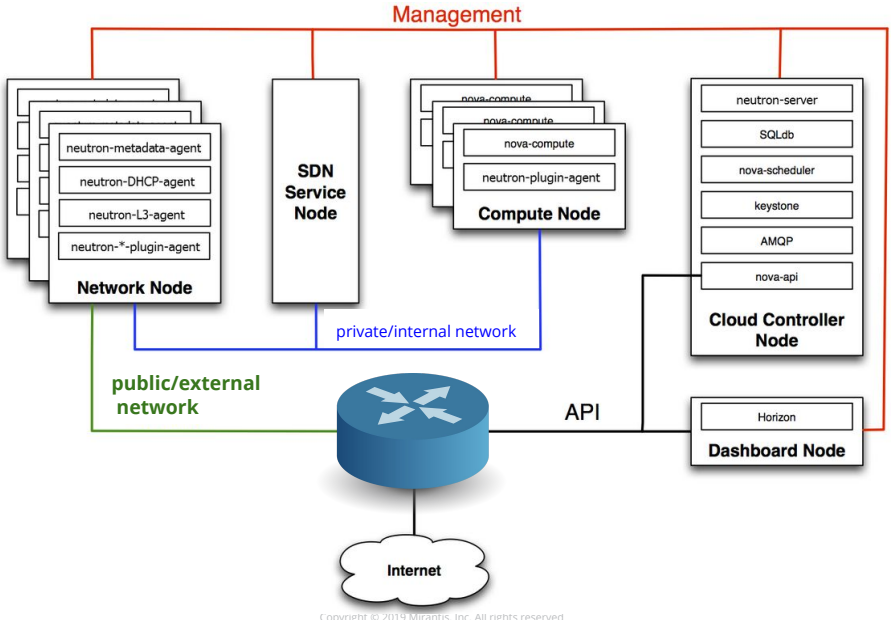
MIRANTIS

9

Data plane:

- Core services required to maintain availability of running instances, networks, storage, and additional services running on top of those resources.
- These services are often referred to as the Data Plane services, and are generally expected to be available all the time.
- *Storage network* is 1 example of a data plane service
  - Hosts traffic between the VMs and their application datasets that are on external storage systems.
- Other examples: Disk arrays, instances, OVS bridges

OpenStack networks - big picture (second view)



MIRANTIS

10

This slide looks at the OpenStack networks from a different viewpoint.
In this view, you can see:

- The Network node provides the connection to the external/public network
- The Management Network connects the nodes running OpenStack. In this case, the Controller, Compute, and Network nodes.
- The API network allows the OpenStack processes, running on a Controller node, to be accessed from the Internet.
- The private/internal network connects the Network and Compute nodes; public/external connectivity provided through the network node.
- network provider services (SDN server/services): Provides additional networking services to tenant networks. These SDN services may interact with *neutron-server*, *neutron-plugin*, and plugin-agents through communication channels such as REST APIs.

For more information on the OpenStack networking architecture:
https://docs.openstack.org/security-guide/networking/architecture.html

OpenStack Networking: Recap

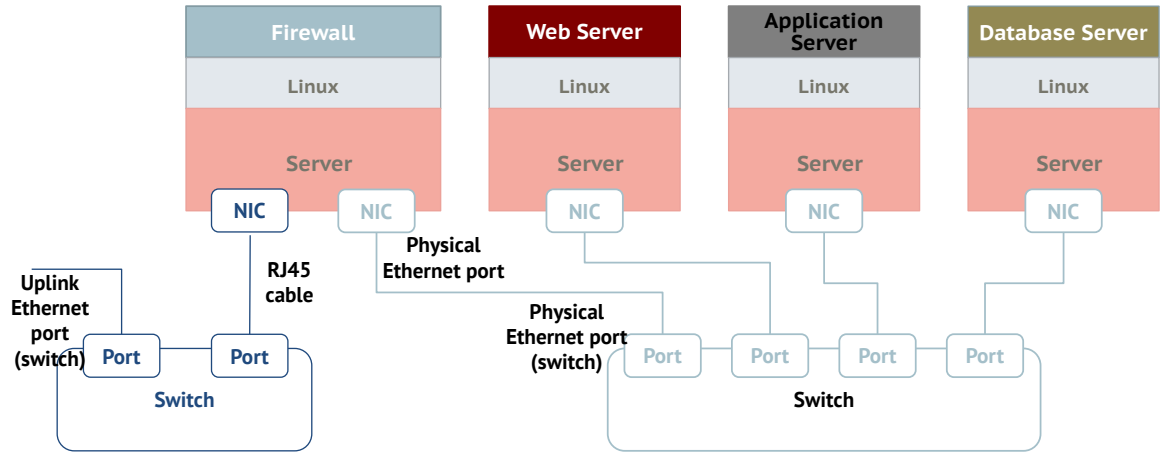
- An OpenStack installation may include the following types of networks:
  - Management network
  - Storage network
  - Internal (private/guest) network
  - External (public) network
  - API network
- Neutron concerned with private and external networks
  - But cloud installer must consider security and bandwidth requirements for all the networks

Network virtualization

Implementation constructs

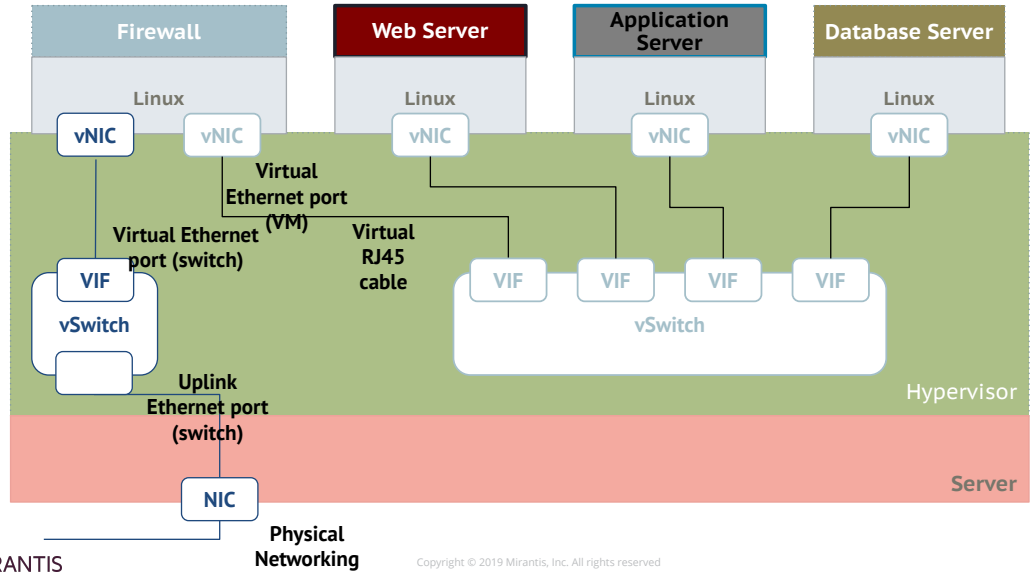
Networking in the small

Traditional networking infrastructure



In a traditional networking infrastructure, each server (firewall, web, application, database) connects to a physical switch through a physical NIC (network interface card).

Virtualized networking infrastructure



Virtualization is nothing but abstracting operating system, application, storage or network away from the true underlying hardware or software. It creates the illusion of physical hardware to achieve the goal of operating system isolation.
Besides virtualizing a server, many network devices must be virtualized to fully map a traditional environment to an equivalent virtual environment. These devices include virtual nics, switches and ethernet cables.

A hypervisor is computer software, firmware or hardware that creates and runs virtual machines. A computer on which a hypervisor runs one or more virtual machines is called a *host machine*, and each virtual machine is called a *guest machine*.

Device Virtualization

vNIC: I/O SW-based sharing

- VM guest writes packets to a file descriptor (FD)
- Hypervisor reads from FD and passes to host networking stack (virtual or physical)
- Device emulation – full virtualization
  - Mimics widely supported real devices (such as an Intel 1Gb NIC) and utilize existing drivers in the guest OS
  - I/O operations have to traverse two I/O stacks, one in the VM and one in the hypervisor
  - For example, VMware ESX, Hyper-V, Xen
- Split-driver model – para-virtualization
  - Uses a front-end driver in the guest VM that works together with a back-end driver in the hypervisor
  - No need to emulate an entire device
  - For example, KVM/QEMU, Virtual Box

A virtual network interface card (vNIC) represents the configuration of a VM connected to a network. A VM can be configured to have multiple vNICs. When a VM is provisioned, each of its associated vNICs can be attached to a virtual network bridge (virtual switch) in order to gain connectivity to a specified network.

The distinction in hypervisor type (full versus para-virtualization) depends on whether or not the hypervisor requires an operating system.

Full virtualization is also known as *type-1* virtualization. With full virtualization, the hypervisor is deployed directly on a host. No operating system required. The hypervisor provides the operating system. Applications running on a fully virtualized VM are unaware of the virtualization.

Para-virtualization is also known as *type-2* virtualization. With para-virtualization, an operating system and hypervisor are installed on a physical server (host). Software and hardware (virtual) in the VMs are aware they are virtualized and are usually running tools such as virtio to supplement the virtual environment.

TAP

- Virtual network interface (kernel device)
- Operates on the Ethernet frame level (L2)

- Virtual ethernet device, connects VM (guest) to virtual switches (Linux bridge or Open vSwitch)
  - For example, VM connects to OVS *integration bridge* using TAP

Open vSwitch and Linux Bridge are discussed the the *Neutron Networking: Deep Dive* lecture.
In Neutron, the TAP interface represents a Neutron port and is allocated to a VM instance during provisioning of the VM. A TAP network interface is managed by a user process within the VM:

- when an Ethernet frame is sent to the network interface, the user process receives this Ethernet frame;
- the user process can send Ethernet frames to this network interface.

virtual Ethernet (vEth) pair

- virtual Ethernet device (veth)
  - Virtual RJ45 cable
  - Created in pairs of virtual interfaces
  - Sometimes called *patch ports*
- If a packet is sent to one interface, it will come out the other interface (behaves like a tunnel)
- OpenStack: Connects two Open vSwitch bridges together

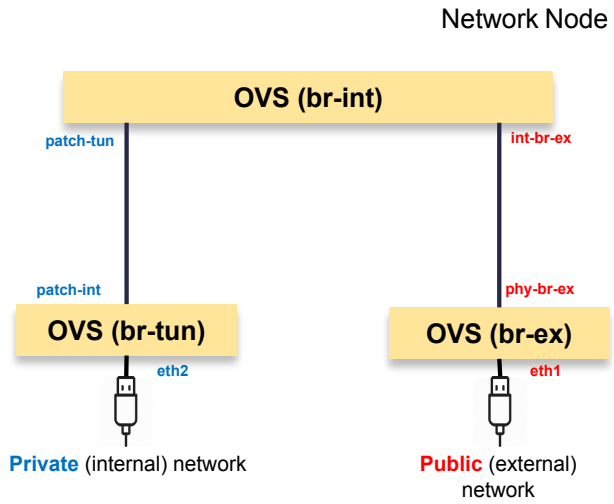
http://stackalytics.com/report/blueprint/neutron/openvswitch-patch-port-use

virtual Ethernet (vEth) pair: OVS example

```
Bridge br-tun
  Port patch-int
    Interface patch-int
      type: patch
      options: {peer=patch-tun}
...

Bridge br-int
  Port patch-tun
    Interface patch-tun
      type: patch
      options: {peer=patch-int}
...
  Port int-br-ex
    Interface int-br-ex
      type: patch
      options: {peer=phy-br-ex}
...

Bridge br-ex
  Port phy-br-ex
    Interface phy-br-ex
      type: patch
      options: {peer=int-br-ex}
```



This slide shows an example with Open vSwitch bridges (vSwitches) on an OpenStack network node. On the left, is the response to an **ovs-vsctl show** command. Notice the 3 bridges (vswitches): br-tun, br-int,and br-ex. For eachm you can see the vEth ports as well as their peer ports. The diagram on the righ shows the actual connections.

VLAN interface

- VLAN interface
  - Add/remove an 802.1Q tag to/from the packet

MIRANTIS

From Wikipedia:
802.1Q, often referred to as **Dot1q**, is the networking standard that supports virtual LANs (VLANs) on an IEEE 802.3 Ethernet network. The standard defines a system of **VLAN tagging** for Ethernet frames and the accompanying procedures to be used by bridges and switches in handling such frames.

Switch virtualization

Linux Bridge

What is a switch?

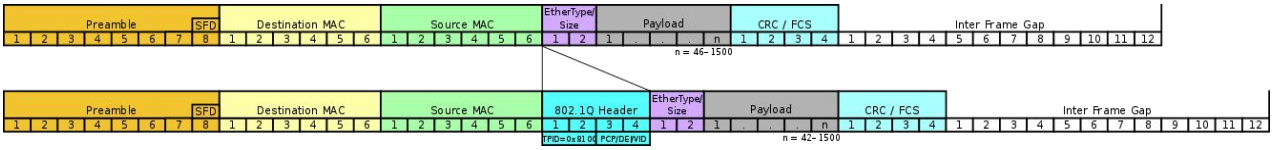
- A network switch is:
  - Device that connects devices together on a computer network
  - Uses packet switching to receive, process, and forward data to the destination device
  - Uses hardware addresses (MAC address) to process and forward data at the Data link layer (L2 of OSI model)
- L2 provides a reliable link between two directly connected nodes, by detecting and possibly correcting errors that may occur in the physical layer

Traditional (ANSI/IEEE 802.1D) switch

- Learns packet source MAC address
- Forwards the traffic to known MAC addresses
- If destination is not known, floods on all the ports except incoming – everybody gets it
- Tries to prevent packets collisions

VLAN (ANSI/IEEE 802.1Q) switch

- Implements virtual LANs on Ethernet networks
  - Using 32-bits (4 bytes) - 12-bits of which are for vlan IDs
  - Other bits for quality-of-service prioritization (IEEE 802.1p)
- Untagged frames treated as native-vlan
  - By default, vlan ID 1



802.1Q adds a 32-bit field between the source MAC address and the EtherType fields of the original frame. The minimum frame size is left unchanged at 64 bytes. The maximum frame size is extended from 1,518 bytes to 1,522 bytes.

- The first two bytes are used for the *tag protocol identifier* (**TPID**).
- The other two bytes are used for *tag control information* (**TCI**). The TCI field is further divided into PCP (priority code point), DEI (drop eligible indicator), and VID (VLAN identifier).

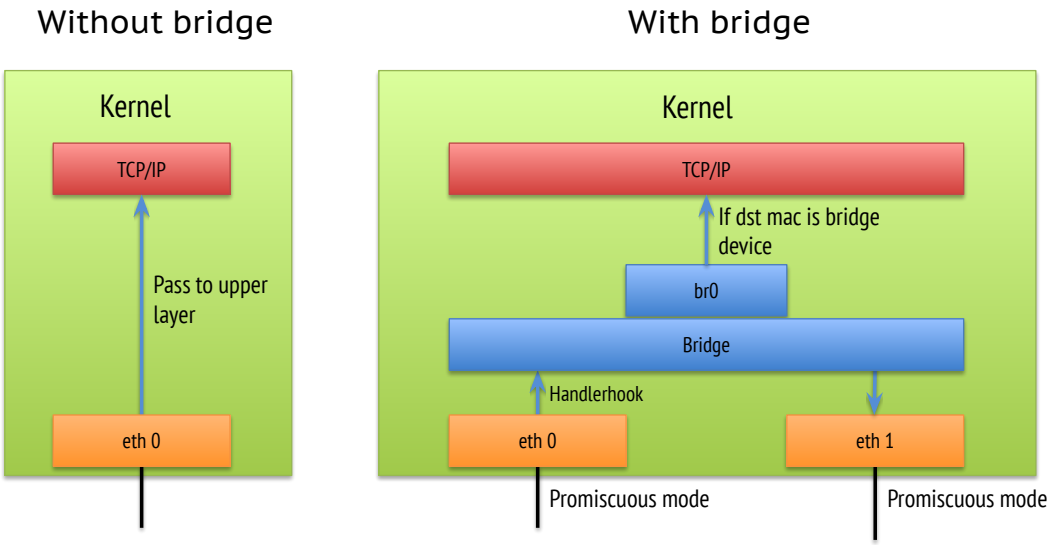
https://en.wikipedia.org/wiki/IEEE_802.1Q
QOS: https://en.wikipedia.org/wiki/IEEE_P802.1p
SPB: https://en.wikipedia.org/wiki/IEEE_802.1aq

Linux Bridge

- Open source L2 virtual switch for Linux
  - connects more than one LAN segment at Layer-2
- Implements subset of 802.1D standard
  - Supports forwarding database (FDB), spanning tree protocol (STP)
- Integrated into kernel 2.4
- Can be created:
  - In network configuration files
  - Or using **brctl** CLI:
    - brctl addbr br0
    - brctl addif br0 eth0

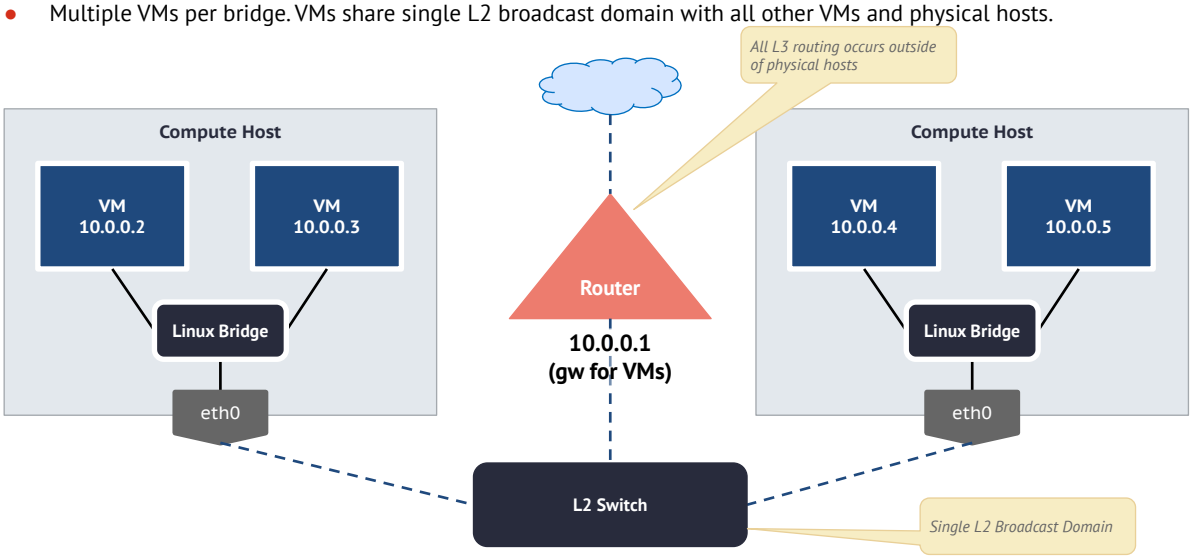
Even though it is called a bridge, *the Linux bridge is really a virtual switch* and used with KVM/QEMU hypervisor. Linux Bridge is a kernel module. Linux Bridge is administered using **brctl** command on Linux.
A Linux bridge behaves like a network switch. It forwards packets between interfaces that are connected to it. It's usually used for forwarding packets on routers, on gateways, or **between VMs and network namespaces** on a host. It also supports STP, VLAN filter, and multicast snooping.

Linux Bridge



Linux Bridge example

- Single Linux bridge per physical host, attached to a single physical NIC (eth0).
- Multiple VMs per bridge. VMs share single L2 broadcast domain with all other VMs and physical hosts.



Before there was OpenStack, there was the Linux Bridge.
It could be used to bridge VMs together in a virtual layer-2 segment, and optionally bridge them to a physical interface, connected to a physical switch – and from there to a router. A Linux bridge is a layer 2 device that allows multiple ethernet segments to be connected together

Switch virtualization

Open vSwitch (OVS)

Open vSwitch (OVS)

- Open source multi-layer (L2-L4) virtual switch
- Released in 2011, Integrated into kernel 2.6.32,
- Includes kernel and user space components
- Supports:
  - port mirroring and link aggregation
  - Per VM interface traffic policing, Fine-grained QoS control
  - IEEE 802.1D source learning (FDB)
  - IEEE 802.1Q VLAN
  - OpenFlow protocol support (extensions for virtualization)
  - Multiple overlay protocols (VxLAN, GRE, IPsec, STT, GENEVE)

Open vSwitch For Windows

- Open vSwitch 2.5
  - Uses Hyper-V datapath for kernel portion
- Hyper-V virtual switch
  - L2 switch
  - Supports IEEE 802.1Q

http://superuser.openstack.org/articles/tutorial-open-vswitch-hyper-v-openstack/
https://cloudbase.it/installing-openstack-nova-compute-on-hyper-v/

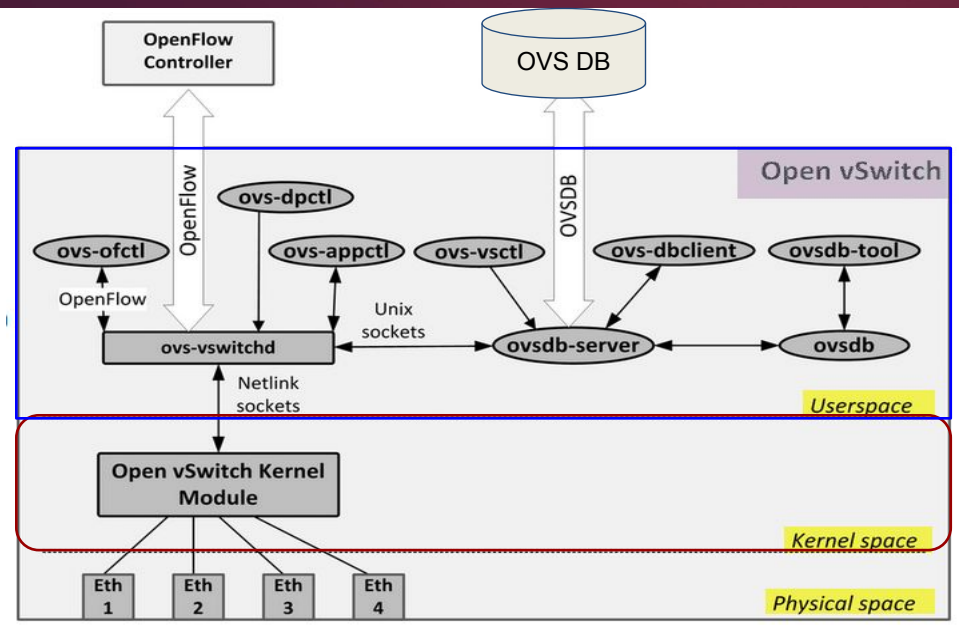
Open vSwitch components

- `ovs-vswitchd`: daemon that implements the switch, along with a companion Linux kernel module for flow-based switching
- `ovs-vsctl`: utility to query/update ovs-vswitchd config
- `ovs-ofctl`: utility to query/control OpenFlow switches and controllers
- `ovs-appctl`: a utility that sends commands to running Open vSwitch daemons
- `ovs-dpctl`: tool for configuring the switch kernel module
- `ovsdb-server`: a lightweight database server that ovs-vswitchd queries to obtain its configuration

From an OpenStack perspective, you might use **ovs-vsctl** to manage the OVS vSwitches and **ovs-ofctl** to manage packet filtering rules.

For more information on Open vSwitch:
http://docs.openvswitch.org/en/latest/intro/what-is-ovs/

OVS architecture



This slide illustrates many of the OVS components (ovs-vswitchd, ovs-vsctl, ovs-ofctl), as well as several DB processes. **ovs-vswitchd** is the primary process (daemon) for OVS.

VMs connect to the eth ports in the physical space. The VMs must be running in **promiscuous mode**. In promiscuous mode, a network adapter does not filter packets. Each network packet on the network segment is directly passed to the operating system or a monitoring application. The packet data is also accessible by any virtual machine (VM) or guest operating system running on the host system.

In the labs, you use the **ovs-vsctl** command to create and manage bridges (virtual switches) and their ports (vEth ports).

In an environment using only Open vSwitch, without Linux Bridges, you might also need to become familiar with **ovs-ofctl** and **ovs-dpctl** commands to view the data packets and flows.

Overlay networks

Overlay networking (sometimes called an SDN overlay) is a method of using software to create layers of network abstraction that can be used to run multiple separate, discrete virtualized network layers on top of the physical network, often providing new applications or security benefits.
Network overlays can dramatically increase the number of virtual subnets that can be created on a physical network, which in turn supports multitenancy and virtualization features. In other words, the number of traditional VLANs can be exhausted quickly.

VLAN versus overlay networks scalability

- VLAN networks have a restriction of 4096 VLAN IDs
  - Problem for many cloud implementations, especially service providers

NETWORK TYPE	VLAN ID NUMBER OF BITS	MAX NUMBER OF TENANT NETWORKS
VLAN	12	4096
VXLAN	24	16,777,216
GRE	32	4,294,967,296
STT	64	Virtually unlimited

- Solution: L2 over L3 tunneling
  - Segments the network using *L3 packets*

VXLAN: Virtual Extensible LAN
GRE: Generic Routing Encapsulation (defined by RFC 2784.)
STT: Stateless Transport Tunneling Protocol

All 3 protocols work by encapsulating a payload -- that is, an inner packet that needs to be delivered to a destination network -- inside an outer IP packet. The tunnel endpoints send payloads through the tunnels by *routing encapsulated packets* through intervening IP networks. Other IP routers along the way do not parse the payload (the inner packet); they only parse the outer IP packet as they forward it towards the tunnel endpoint.

There are other tunnel protocols (not discussed in this lecture), such as:

- NVGRE (Network Virtualization using Generic Routing Encapsulation)
- GENEVE (Generic Network Virtualization Encapsulation)

What is VXLAN?

- As described in RFC 7348:

  Virtual eXtensible Local Area Network (VXLAN) … is used to address the need for overlay networks within virtualized data centers accommodating multiple tenants. The scheme and the related protocols can be used in networks for cloud service providers and enterprise data centers.

From Wikipedia:
Virtual Extensible LAN (VXLAN) is a network virtualization technology that attempts to address the scalability problems associated with large cloud computing deployments. It uses a VLAN-like encapsulation technique to encapsulate OSI layer 2 Ethernet frames within layer 4 UDP datagrams, using 4789 as the default IANA-assigned destination UDP port number. VXLAN endpoints, which terminate VXLAN tunnels and may be either virtual or physical switch ports, are known as VXLAN tunnel endpoints (VTEPs).
https://en.wikipedia.org/wiki/Virtual_Extensible_LAN

The full RFC is here:
https://tools.ietf.org/html/rfc7348

What is VXLAN?

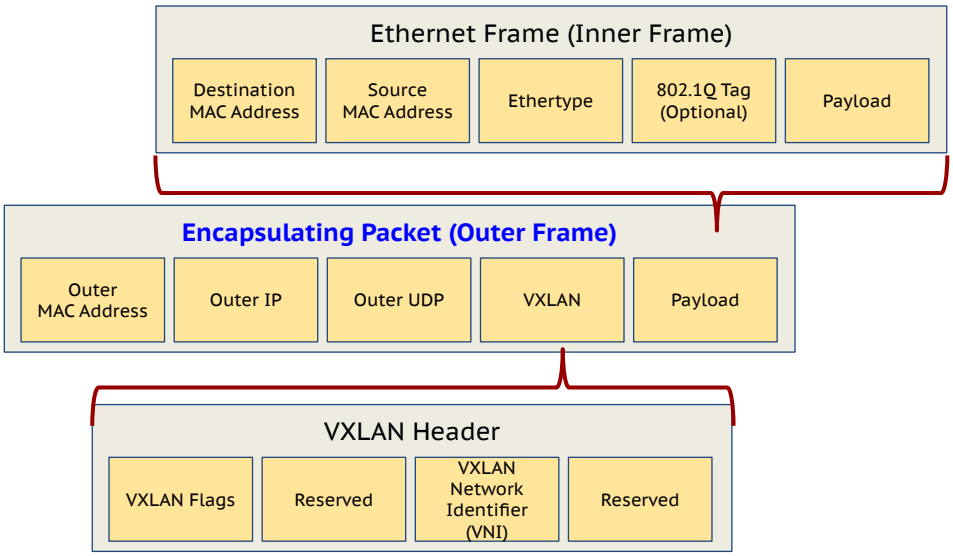
- *Tunneling protocol* initially developed by Cisco, Arista, VMware, and others
- Lays on top of the physical network - also known as a *network overlay*
- Designed to use existing networking infrastructure
- Expands a single layer 2 network over layer 3
- Ideal for virtualized environments
  - Allows for dynamic changes to be performed programmatically
  - Underlying physical network need not be changed/re-configured

Why VXLAN?

- Addresses limitations of STP and VLAN ranges
  - 12-bit field for VLAN sets upper limit of tenant networks to 4096
- Designed from the ground up with multi-tenant environments in mind
- Addresses inadequate table sizes at top of the rack (ToR) switch
  - In virtualized environments, ToR switch(es) need to maintain table entries for MAC addresses for all physical and virtual servers
  - If tables overflow, switch may stop learning addresses causing significant flooding of subsequent frames

https://tools.ietf.org/html/rfc7348#section-3

Layer 2 over layer 3



Outer MAC Address: The outer destination MAC address in this frame may be the address of the target VTEP or of an intermediate Layer 3 router.

Outer IP Header: This is the outer IP header with the source IP address indicating the IP address of the VTEP over which the communicating VM (as represented by the inner source MAC address) is running. The destination IP address can be a unicast or multicast IP address (see Sections 4.1 and 4.2). When it is a unicast IP address, it represents the IP address of the VTEP connecting the communicating VM as represented by the inner destination MAC address. For multicast destination IP addresses, please refer to the scenarios detailed in Section 4.2.

Outer UDP Header: This is the outer UDP header with a source port provided by the VTEP and the destination port being a well-known UDP port.

- Destination Port: IANA has assigned the value 4789 for the VXLAN UDP port, and this value SHOULD be used by default as the destination UDP port. Some early implementations of VXLAN have used other values for the destination port. To enable interoperability with these implementations, the destination port SHOULD be configurable.
- Source Port: It is recommended that the UDP source port number be calculated using a hash of fields from the inner packet -- one example being a hash of the inner Ethernet frame's headers. This is to enable a level of entropy for the ECMP/load-balancing of the VM-to-VM traffic across the VXLAN overlay. When calculating the UDP source port number in this manner, it is RECOMMENDED that the value be in the dynamic/private port range 49152-65535 [RFC6335].
- UDP Checksum: It SHOULD be transmitted as zero. When a packet is received with a UDP checksum of zero, it MUST be accepted for decapsulation. Optionally, if the encapsulating end point includes a non-zero UDP checksum, it MUST be correctly calculated across the entire packet including the IP header, UDP header, VXLAN header, and encapsulated MAC frame. When a decapsulating end point receives a packet with a non-zero checksum, it MAY choose to verify the checksum value. If it chooses to perform such verification, and the verification fails, the packet MUST be dropped. If the decapsulating destination chooses not to perform the verification, or performs it successfully, the packet MUST be accepted for decapsulation.

VXLAN Header: This is an 8-byte field that has:

- Flags (8 bits): where the I flag MUST be set to 1 for a valid VXLAN Network ID (VNI). The other 7 bits (designated "R") are reserved fields and MUST be set to zero on transmission and ignored on receipt.

- VXLAN Segment ID/VXLAN Network Identifier (VNI): this is a 24-bit value used to designate the individual VXLAN overlay network on which the communicating VMs are situated. VMs in different VXLAN overlay networks cannot communicate with each other.
- Reserved fields (24 bits and 8 bits): MUST be set to zero on transmission and ignored on receipt.

Summary

You should now be able to:

- Describe the 5 networks that *might be seen* in an OpenStack deployment
- Describe available options for switch virtualization
- Describe supported network protocols, including options using overlay networks (*tunneling*)

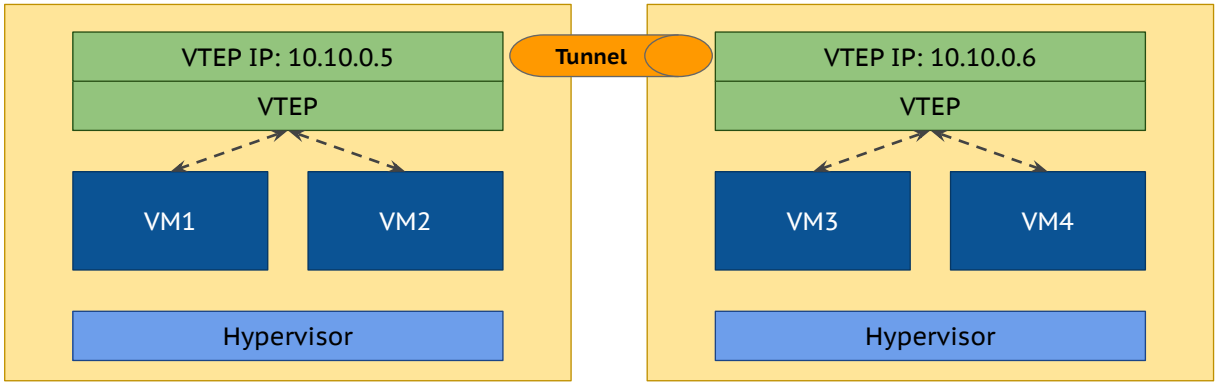
Reference

Virtual Tunnel End Point (VTEP)

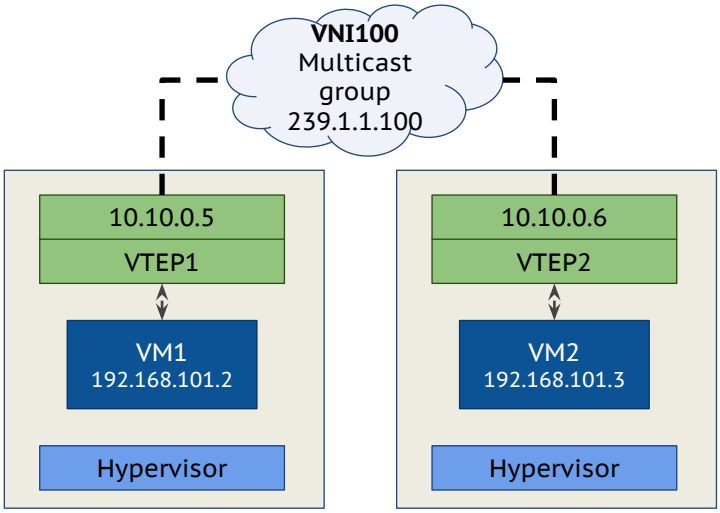
Key concepts

- Virtual Tunnel End Point (VTEP)
  - Network device that corresponds to the beginning or end of the tunnel
  - Owns IP address that will be used in network communication
  - Can reside on a virtual switch, physical switch, firewalls, routers, etc.
  - Responsible for performing encapsulation and decapsulation packets for the different VNIs
- Virtual Network Identifier (VNI)
  - 24 bit field used to identify a particular VXLAN segment
  - Analogous to a VLAN ID in the traditional networking world

How does it work?



How does it work?



- VM1 doesn't know the MAC address for VM2 yet - sends an ARP broadcast packet
- VTEP1 receives ARP packet
- VTEP1 encapsulates packet into a multicast packet to group 239.1.1.100
- All VTEPs subscribed to the multicast group receive the packet
- Each VTEP decapsulates the packet to look at VNI the packet was placed on (VNI100)
- If that VNI is in use on that system, the VTEP forwards the packet to VMs on that VNI
- VTEP2 creates a record in local VXLAN table mapping VM1's MAC to VTEP1 (10.10.0.5)
- VM2 receives ARP packet and sends back a response to VM1
- VTEP2 encapsulates response and sends back to VTEP1 (10.10.0.5)
- VTEP1 receives response, decapsulates the packet and forwards to VM1
- VTEP1 creates a record in local VXLAN table mapping VM2's MAC to VTEP2 (10.10.0.6)

MAC learning behavior

- When VMs on different nodes look to communicate, it is up to the VTEPs to handle the communication:
  - If the VTEP has an entry for the destination MAC in its table, then it will pass the packet to the VTEP associated with that MAC
  - If the VTEP does not have an entry for the destination MAC in its table, then it will flood the packet to all VTEPs in the multicast group
- Flooding is expensive and does not scale well