Chapter 19 :

Basic Networking Concepts & Neutron

- Introduction
- Subnets & CIDR
- IP tables
- Iproute2 commands
- Encapsulation – VLAN, VXLAN, GRE
- How a VM get an IP?

19. Basic Networking Concepts & Neutron

In this chapter we will cover:

1: Basic Networking Concepts
2: Iptables
3:Iproute2 Commands
4: Encapsulation in Neutron
5:How a VM gets an IP

1. Basic Networking Concepts

Ethernet

Ethernet is the most widely installed local area network (LAN) technology. Ethernet is a *link layer* protocol in the TCP/IP stack, describing how networked devices can format data for transmission to other network devices on the same network segment, and how to put that data out on the network connection. It touches both Layer 1 (the physical layer) and Layer 2 (the data link layer) on the OSI network protocol model. Ethernet defines two units of transmission, packet and frame. The frame includes not just the "payload" of data being transmitted but also addressing information identifying the physical "Media Access Control" (MAC) addresses of both sender and receiver, VLAN tagging and quality of service information, and error-correction information to detect problems in transmission. Each frame is wrapped in a packet, which affixes several bytes of information used in establishing the connection and marking where the frame starts.

In particular, in an OpenStack environment, every virtual machine instance has a unique MAC address, which is different from the MAC address of the compute host. A MAC address has 48 bits and is typically represented as a hexadecimal string, such as 08:00:27:b9:88:74. You can think of an Ethernet network as a single bus that each of the network hosts connects to. In Modern Ethernet networks use switches to interconnect the network hosts. A switch is a box of networking hardware with a large number of ports, that forwards Ethernet frames from one connected host to another. When hosts first send frames over the switch, the switch doesn’t know which MAC address is associated with which port. If an Ethernet frame is destined for an unknown MAC address, the switch broadcasts the frame to all ports. The port learns which MAC addresses are at which ports by observing the traffic. Once it knows which MAC address is associated with a port, it can send Ethernet frames to the correct port instead of broadcasting.

IP

The Internet Protocol (IP) specifies how to route packets between hosts that are connected to different local networks. IP relies on special network hosts called *routers* or *gateways*. A router is a host that is connected to at least two local networks and can forward IP packets from one local network to another. A router has multiple IP addresses: one for each of the networks it is connected to.

In the OSI model of networking protocols, IP occupies the third layer, which is known as the network layer. When discussing IP, you will often hear terms such as *layer 3*, *L3*, and *network layer*.

A host sending a packet to an IP address consults its *routing table* to determine which machine on the local network(s) the packet should be sent to. The routing table maintains a list of the subnets associated with each local network that the host is directly connected to, as well as a list of routers that are on these local networks.

VLANs

A virtual local area network (**VLAN**) is any broadcast domain that is partitioned and isolated in a computer network at the data link layer (OSI layer 2). To subdivide a network into virtual LANs, one configures a network switch or router. It is a networking technology that enables a single switch to act as if it was multiple independent switches. Specifically, two hosts that are connected to the same switch but on different VLANs do not see each other's traffic. This is handy because there may be situations where you need the functionality of multiple parallel physical networks but you'd rather not spend the money on buying parallel hardware.

OpenStack is able to take advantage of VLANs to isolate the traffic of different tenants, even if the tenants happen to have instances running on the same compute host. Each VLAN has an associated numerical ID, between 1 and 4095. For instance "VLAN 15" to refer to the VLAN with numerical ID of 15.

Subnets and ARP

NICs use MAC addresses to address network hosts, TCP/IP applications use IP addresses. The Address Resolution Protocol (ARP) bridges the gap between Ethernet and IP by translating IP addresses into MAC addresses.

IP addresses are broken up into two parts: a *network number* and a *host identifier*. Two hosts can only communicate directly over Ethernet if they are on the same local network. ARP assumes that all machines that are in the same subnet are on the same local network. Network administrators must take care when assigning IP addresses and netmasks to hosts so that any two

hosts that are in the same subnet are on the same local network, otherwise ARP does not work properly.

The process of dividing a network into smaller network sections is called **subnetting**. This can be useful for many different purposes and helps isolate groups of hosts together and deal with them easily. Each address space is divided into a network portion and a host portion. The amount the address that each of these take up is dependent on the class that the address belongs to. For instance, for class C addresses, the first 3 octets are used to describe the network. For the address192.168.0.15, the 192.168.0 portion describes the network and the 15 describes the host. A Subnet mask is a 32-bit number that masks an IP address, and divides the IP address into network address and host address. Subnet Mask is made by setting network bits to all "1"s and setting host bits to all "0"s.

To calculate the network number of an IP address, you must know the *netmask* associated with the address. A netmask indicates how many of the bits in the 32-bit IP address make up the network number.

There are two syntaxes for expressing a netmask:

- dotted quad
- classless inter-domain routing (CIDR)

Consider an IP address of 192.168.1.5, where the first 24 bits of the address are the network number. In dotted quad notation, the netmask would be written as 255.255.255.0.

Classless Inter-Domain Routing (CIDR), was developed as an alternative to traditional subnetting. The idea is that you can add a specification in the IP address itself as to the number of significant bits that make up the routing or networking portion. For example, we could express the idea that the IP address 192.168.0.15 is associated with the netmask 255.255.255.0 by using the CIDR notation of 192.168.0.15/24. This means that the first 24 bits of the IP address given are considered significant for the network routing.

This allows us some interesting possibilities. We can use these to reference "supernets". In this case, we mean a more inclusive address range that is not possible with a traditional subnet mask. For instance, in a class C network, like above, we could not combine the addresses from the networks 192.168.0.0 and 192.168.1.0 because the netmask for class C addresses is 255.255.255.0. However, using CIDR notation, we can combine these blocks by referencing this chunk as192.168.0.0/23. This specifies that there are 23 bits used for the network portion that we are referring to.

2. Iptables

Firewall decides fate of packets incoming and outgoing in system. Iptables is an extremely flexible firewall utility built for Linux operating systems. Whether you're a novice Linux geek or a system administrator, there's probably some way that iptables can be a great use to you.

iptables is a user-space application program that allows a system administrator to configure the tables provided by the Linux kernel firewall (implemented as different Netfilter modules) and the chains and rules it stores. Different kernel modules and programs are currently used for different protocols; *iptables* applies to IPv4, *ip6tables* to IPv6, *arptables* to ARP, and *ebtables* to Ethernet frames.

Basically, Iptable is a user space tool used to add and remove rules to the kernel level firewall. It needs to be called every time you want to add a rule. iptables requires elevated privileges to operate and must be executed by user root, otherwise it fails to function. On most Linux systems, iptables is installed as /usr/sbin/iptables and documented in its man pages, which can be opened using `man iptables` when installed. It may also be found in /sbin/iptables, but since iptables is more like a service rather than an "essential binary", the preferred location remains /usr/sbin.

The iptables firewall operates by comparing network traffic against a set of **rules**. The rules define the characteristics that a packet must have to match the rule, and the action that should be taken for matching packets.

There are many options to establish which packets match a specific rule. You can match the packet protocol type, the source or destination address or port, the interface that is being used, its relation to previous packets, etc.

When the defined pattern matches, the action that takes place is called a **target**. A target can be a final policy decision for the packet, such as accept, or drop. It can also be move the packet to a different chain for processing, or simply log the encounter. There are many options.

These rules are organized into groups called **chains**. A chain is a set of rules that a packet is checked against sequentially. When the packet matches one of the rules, it executes the associated action and is not checked against the remaining rules in the chain.

A user can create chains as needed. There are three chains defined by default. They are:

- **INPUT**: This chain handles all packets that are addressed to your server.
- **OUTPUT**: This chain contains rules for traffic created by your server.
- **FORWARD**: This chain is used to deal with traffic destined for other servers that are not created on your server. This chain is basically a way to configure your server to route requests to other machines.

Each chain can contain zero or more rules, and has a default **policy**. The policy determines what happens when a packet drops through all of the rules in the chain and does not match any rule. You can either drop the packet or accept the packet if no rules match.

The iptables commands are as follows:

- -A — Appends the iptables rule to the end of the specified chain. This is the command used to add a rule when rule order in the chain does not matter.
- -C — Checks a particular rule before adding it to the user-specified chain. This command can help you construct complicated iptables rules by prompting you for additional parameters and options.
- -D — Deletes a rule in a particular chain by number (such as 5 for the fifth rule in a chain). You can also type the entire rule, and iptables deletes the rule in the chain that matches it.
- -E — Renames a user-defined chain. This does not affect the structure of the table.
- -F — Flushes the selected chain, which effectively deletes every rule in the the chain. If no chain is specified, this command flushes every rule from every chain.
- -h — Provides a list of command structures, as well as a quick summary of command parameters and options.
- -I — Inserts a rule in a chain at a point specified by a user-defined integer value. If no number is specified, iptables places the command at the top of the chain.

3. Iproute2 commands

iproute2 is a collection of userspace utilities used to communicate with various Linux kernel components over the netlink protocol. More specifically, the iproute2 utilities are used for controlling TCP and UDP IP networking and traffic control in the Linux kernel, in both IPv4 and IPv6 networks, as well as for configuring the device drivers for network interface controllers (NICs) and wireless network interface controllers (WNICs).

Configuration utilities replaced by iproute2 are shown in the following table:

Purpose	Legacy utility	iproute2 equivalent
Address and link configuration	ifconfig	ip addr, ip link
Routing tables	route	ip route
Neighbors	arp	ip neigh
VLAN	vconfig	ip link
Tunnels	iptunnel	ip tunnel
Bridges	brctl	ip link, bridge
Multicast	ipmaddr	ip maddr
Statistics	netstat	ip -s, ss

The ip route get command outputs the route for a destination IP address. From the above example, destination IP address 10.0.2.14 is on the local network of eth0 and would be sent directly:

```
$ ip route get 10.0.2.14

10.0.2.14 dev eth0  src 10.0.2.15
```

The destination IP address 93.184.216.34 is not on any of the connected local networks and would be forwarded to the default gateway at 10.0.2.2:

```
$ ip route get 93.184.216.34

93.184.216.34 via 10.0.2.2 dev eth0  src 10.0.2.15
```

4. Encapsulation:

In computer networking, Encapsulation is a method of designing modular communication protocols in which logically separate functions in the network are abstracted from their underlying structures by inclusion or information hiding within higher level objects.

The physical layer is responsible for physical transmission of the data. Link encapsulation allows local area networking and Internet Protocol (IP) provides global addressing of individual computers; Transmission Control Protocol (TCP) adds application or process selection, i.e., the port specifies the service such as a Web or TFTP server

In simple words, Encapsulation is a process to hide or protect a process from the possibility of outside interference or misuse of the system while simplifying the use of the system itself, also makes one type of network data packets to other data types. Encapsulation occurs when a protocol that is on the lower layer receives data from the protocol that is at a higher layer and put the data into a data format that is understood by the protocol. Access to the internal system so arranged through a set of interfaces.

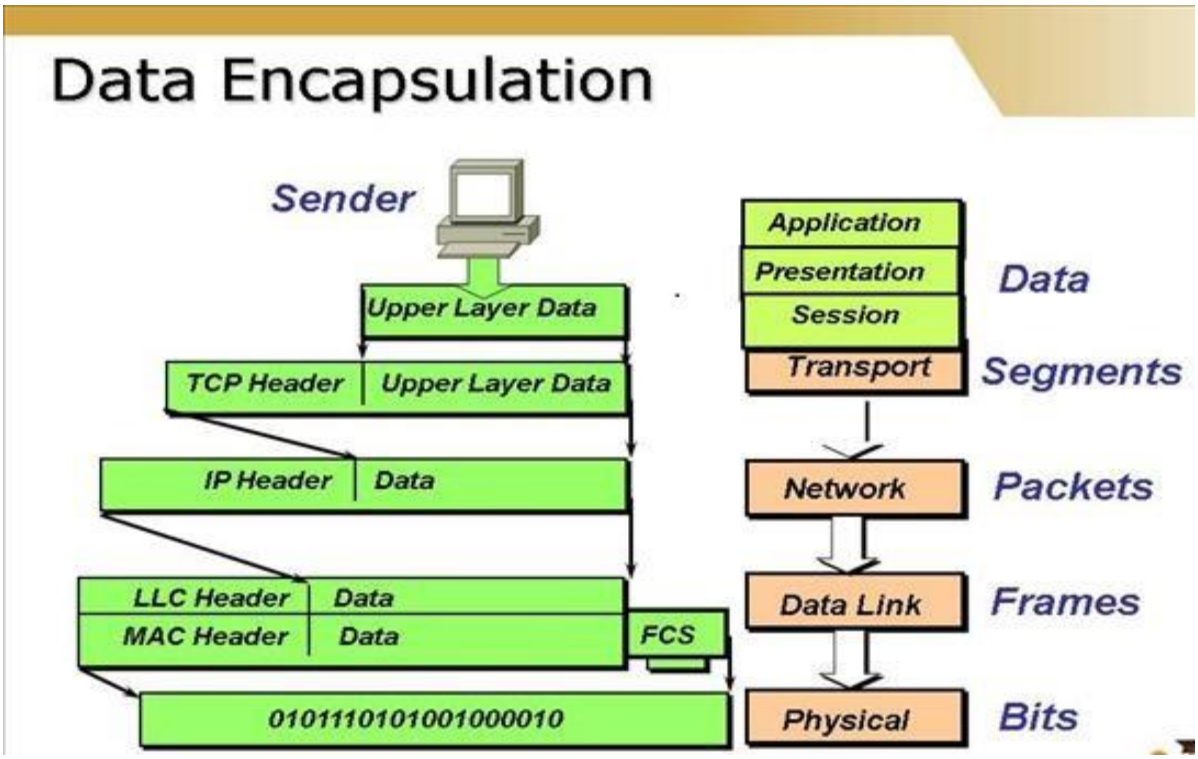

Figure 1: Data Encapsulation Model

Encapsulation is utilized when data is transmitted from the application layer to the physical layer of the OSI reference model. Following the data stream on the figure above:

- Alphanumeric user input is converted to data for transmission on the network (At the upper layers)
- Data is converted to segments, which allow hosts to reliably communicate (At the Transport Layer)
- Segments are converted to packets or datagrams with a source and destination logical address (At the Network layer)
- Packets or datagrams are converted to frames for transmission over an interface to the network (At the Data Link Layer)
- Frames are converted to bits, and uses a synchronization and clocking function (At the Physical Layer)
- Again, data encapsulation occurs when going from layer 7 to layer 1 of the OSI reference model.

About VLAN

VLANs enable efficient traffic separation and provide excellent bandwidth utilization. VLANs also alleviate scaling issues by logically segmenting the physical LAN structure into different subnetworks so that packets are switched only between ports within the same VLAN. This can be very useful for security, broadcast containment, and accounting.

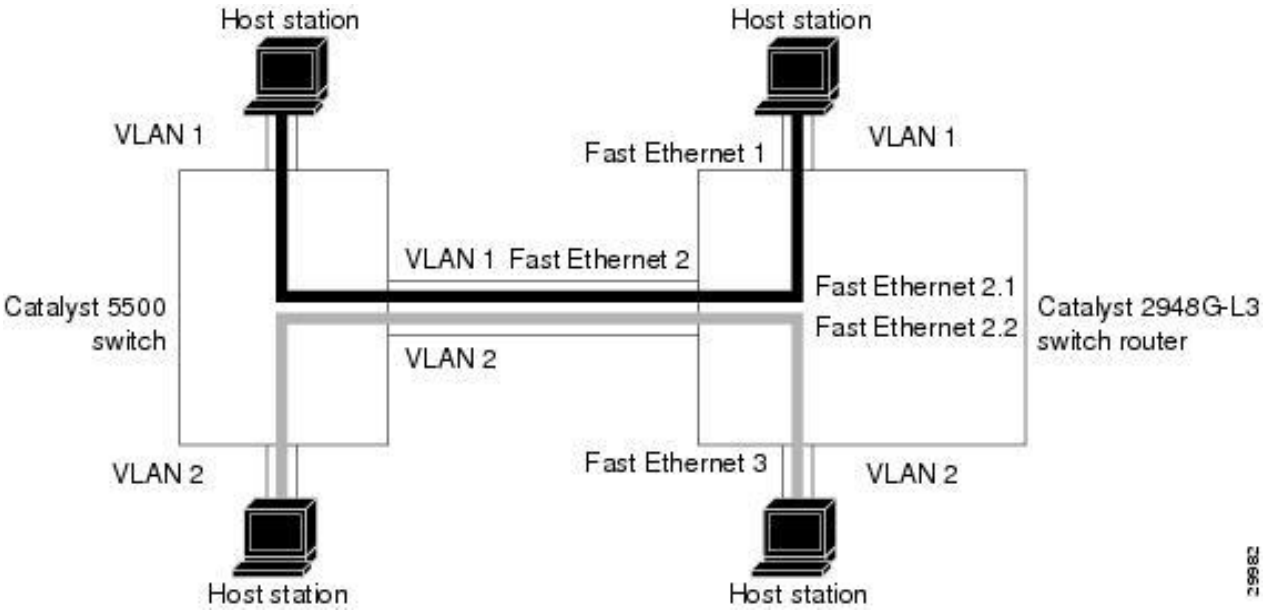

Figure 2: VLANs spanning device in a network

Layer 3 switching software supports a port-based VLAN on a trunk port, which is a port that carries the traffic of multiple VLANs. Each frame transmitted on a trunk link is tagged as belonging to only one VLAN.

Layer 3 switching software supports VLAN frame encapsulation through the Inter-Switch Link (ISL) protocol and the 802.1Q standard on both the Catalyst 2948G-L3 and the Catalyst 4908G-L3 switch routers.

Figure 2 shows a network topology where two VLANs span a Catalyst 5500 switch and a Catalyst 2948G-L3 switch router. Both VLANs in this topology are bridged using the Inter-Switch Link (ISL) protocol.

VXLAN (Virtual Extensible LAN)

VXLAN is a network virtualization technology that attempts to ameliorate the scalability problems associated with large cloud computing deployments. It uses a VLAN-like encapsulation technique to encapsulate MAC-based OSI layer 2 Ethernet frames within layer 4 UDP packets.

Mainly driven by Cisco. The VXLAN packet header includes a 24-bit segment ID (16M unique virtual segments), This ID is usually generated by using pseudo-random source UDP ports (which are hash-generated upon the original MAC addresses within the frame). The incentive is to keep the ordinary 5-tuple based load balancers, to preserve the inter-VM packet order, and it achieve this by "reflecting" the internal packet mac pairs, to a unique UDP port pair. L2 broadcast is converted to IP multicast: VXLAN specifies the use of IP multicast to flood within the virtual segment and relies on dynamic MAC learning. The VXLAN encapsulation increases the size of the packet by 50 bytes, as described below:

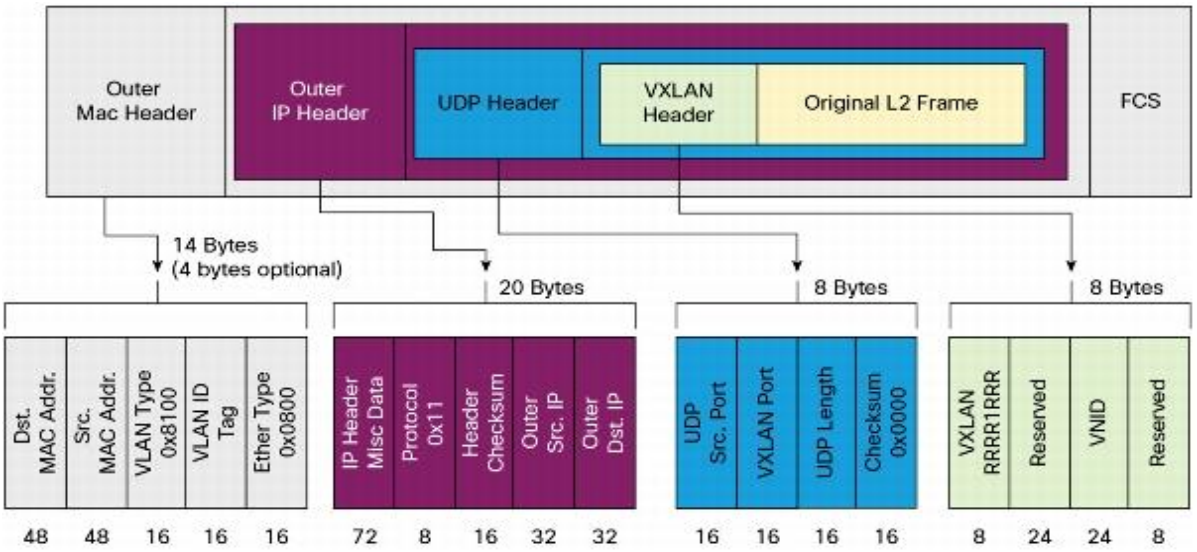

Figure 3: VXLAN Encapsulation

Due to the oversized packets, VXLAN rely on the transport network (to support the increase of the packet size), by requiring it to support jumbo frames.

GRE (Generic Routing Encapsulation)

Generic Routing Encapsulation (GRE) is a protocol that encapsulates packets in order to route other protocols over IP networks.

GRE was developed as a tunneling tool meant to carry any OSI Layer 3 protocol over an IP network. In essence, GRE creates a private point-to-point connection like that of a virtual private network (VPN).

GRE works by encapsulating a payload -- that is, an inner packet that needs to be delivered to a destination network -- inside an outer IP packet. GRE tunnel endpoints send payloads through GRE 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 GRE tunnel endpoint. Upon reaching the tunnel endpoint, GRE encapsulation is removed and the payload is forwarded along to its ultimate destination.

In contrast to IP-to-IP tunneling, GRE tunneling can transport multicast and IPv6 traffic between networks. Advantages of GRE tunnels include the following:

- GRE tunnels encase multiple protocols over a single-protocol backbone.
- GRE tunnels provide workarounds for networks with limited hops.
- GRE tunnels connect discontinuous sub-networks.
- GRE tunnels allow VPNs across wide area networks (WANs).

5. How VM get an IP?

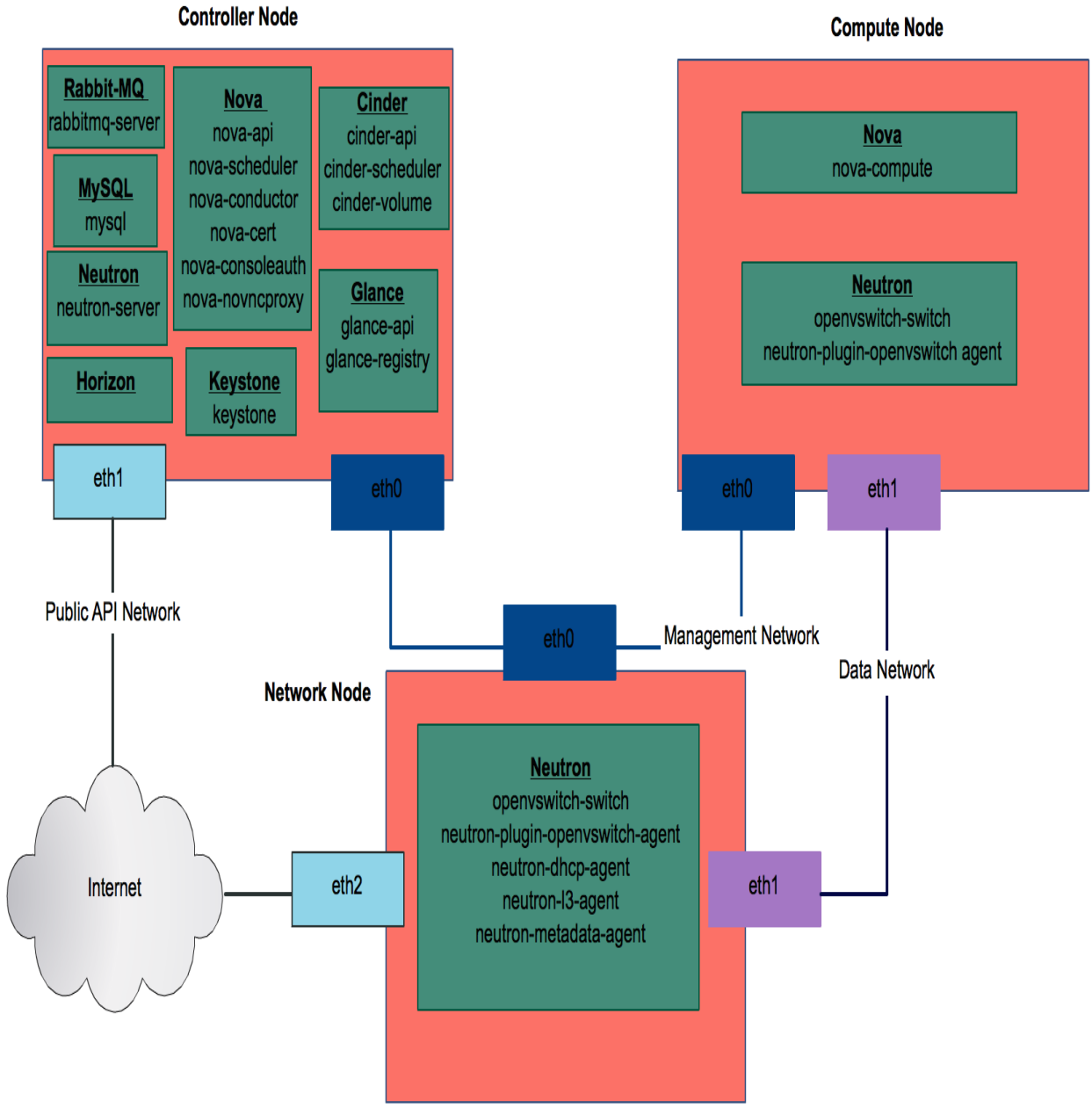

Figure: Neutron Network Connectivity

- **Controller Node** : OpenVswitch sends a call to Nova API at Neutron Server. As a result we create a private network at the **Network Node** (Nothing happens in initial stage before its created). This message don't even leave the node yet.
- This Controller Node creates the subnet : that specifies which network to put it on i.e. here Private. (for an example: 192.168.1.0/24). Now for this we write a command as

```
$ neutron subnet-create --name private-subnet1 private
```

- Now as a result of this, Api call request goes to the Neutron server.
- This Neutron server, not only update the data base, but also put the message into the Rabbit-MQ (that we have a creation of a new server).
- **Network Node**: Services are here to listen Rabbit. DHCP agent will now create NameSpace in Network node

```
$ ip netns #gives all the Name-spaces on the server
```

- Namespace will look like [**q**dhcp ….. number…id]
  Q is from an old name of the neutron & the Id is not the subnet id, but the id of the network.
- For an example Inside the Namespace, if DHCP subnet Id is 192.168.1.0/ => DHCP server id will be 192.168.1.2/ (not 192.168.1.1, as this ip is for the gateway router)
- In the namespace type

```
$ ip a
```

- We will see two interfaces :
  - 1), Package Adaptor &
  - 2) Tap Interface. (that has 192.168.1.2 ip)

  Now this tap-interface is responsible for handling out the IP address.

- Now in the Linux, there exist a program called DNS mask that hands out the IP addresses. So all Neutron DHCP agent will use that DNS mask & binding it with theTap interface IP addresses, now it will hand out that IP addresses on that available range.

- **Network Node:** Now we boot an instance with Nova –boot image & name of the instance. It will give a call to Nova API through Rabbit-MQ, that is in **Controller node.**

- Nova Scheduler takes the request & it finds everyone with Nova Compute node, that's how it picks up the node based on the port number from the request. So that node takes that instance request.

- The instance boots up, so the DHCP has enabled an image that asks for the IP address.

- In between the instance boots up, Nova needs to talk to the Glance, takes the image. Then talks to the Neutron, that API needs to connect to the subnet.

- Now, inside the VM the IP 192.168.1.3 is allocated. So when we created a subnet, instance is attached to the subnet and subnet is attached to the private network

    Connection point b/w instance and subnet is MAC ADDRESS (generated by openstack randomly)
- Now this instance communicate to actual Compute Node with tap interface. This compute node has some port id. That's how we identify which node is the instance from.

- This instance plugs into a bridge inside the Compute Node with the bridge in the private network.

- From here we go forward with some form of encapsulation:

    - VLAN
        - Limitation: We need switches to control the traffic with VLAN
    - GRE (Generic Routing Encapsulation)
    - VXLAN (Virtual Extensible LAN)
        - We can create Jumbo package with it.

- Traffic thus passes through VXLAN that gives the broadcast package. When they passes they add header (eg : vni-100) outside the package. This package will travel outside the node.
- It comes to the Network node & strips off that 100 & goes to the Names space. This Network node sends back 4 times the request & response to the compute node with the outer IP address. That's how it gets an IP.

References:

- http://docs.openstack.org/networking-guide/intro_basic_networking.html#subnets-and-arp
- http://www.howtogeek.com/177621/the-beginners-guide-to-iptables-the-linux-firewall/
- https://en.wikipedia.org/wiki/Iptables

- http://www.ibm.com/developerworks/cloud/library/cl-openstack-network/
- https://en.wikipedia.org/wiki/Iproute2
- http://xmodulo.com/linux-tcpip-networking-net-tools-iproute2.html
- https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/3/html/Reference_Guide/s1-iptables-options.html
- http://www.tecmint.com/basic-guide-on-iptables-linux-firewall-tips-commands/
- https://www.digitalocean.com/community/tutorials/how-the-iptables-firewall-works
- 1 Basic Networking Concepts:
- https://en.wikipedia.org/wiki/Encapsulation_(networking)
- http://searchnetworking.techtarget.com/definition/encapsulation
- http://www.firewall.cx/networking-topics/the-osi-model/179-osi-data-encapsulation.html
- http://computernetworkingsimplified.com/category-1/layering/encapsulation-decapsulation/
- http://studynet-work.blogspot.com/2011/09/encapsulation-and-decapsulation.html
- http://www.tcpipguide.com/free/t_IPDatagramEncapsulation.htm
- https://blogs.vmware.com/cto/geneve-vxlan-network-virtualization-encapsulations/
- http://networkheresy.com/2012/03/04/network-virtualization-encapsulation-and-stateless-tcp-transport-stt/
- https://tools.ietf.org/html/draft-sridharan-virtualization-nvgre-00
- http://www.cisco.com/c/en/us/solutions/collateral/data-center-virtualization/application-centric-infrastructure/white-paper-c11-733127.html
- http://assafmuller.com/2013/10/14/gre-tunnels-in-openstack-neutron/
- http://www.ran-lifshitz.com/2014/08/24/tunneling-and-network-virtualization-nvgre-vxlan/
- http://www.cisco.com/web/JP/event/es/postevent/123/iaf2012/thankyou/docs/5_iaf_VXLAN.pdf
- https://msdn.microsoft.com/en-us/library/windows/hardware/dn144775(v=vs.85).aspx
- http://www.definethecloud.net/nvgre/
- http://blog.ipspace.net/2011/09/nvgre-because-one-standard-just-wouldnt.html
- http://searchsdn.techtarget.com/definition/NVGRE-Network-Virtualization-using-Generic-Routing-Encapsulation
- https://fojta.files.wordpress.com/2012/12/edge-gw-logging.png
- https://www.certificationkits.com/cisco-certification/cisco-ccna-640-802-exam-certification-guide/cisco-ccna-the-osi-model/
- http://www.cisco.com/c/en/us/td/docs/switches/lan/catalyst2948gand4908g/12-0_7_w5_15d/configuration/guide/config/vlan_cnfg.html
- http://searchenterprisewan.techtarget.com/definition/Generic-routing-encapsulation-GRE
- http://utsa.cloudtrain.me/fundamentals/slides/nova.pdf