Module IV

OpenStack Networking

NEUTRON
an OpenStack Community Project

Choice of Connectivity Types and Networking Services

Cloud Computing

Table of contents

OpenStack Networking

- **Network Abstraction:**
- Neutron provides a level of abstraction for networking in OpenStack, allowing users to define and manage networks, subnets, routers, and other networking elements in a virtualized environment.
- **Pluggable Architecture:**
- Neutron has a modular and pluggable architecture, enabling support for various networking technologies and plugins. This flexibility allows organizations to choose the networking solutions that best fit their needs.
- **Multi-Tenancy Support:**
- Neutron is designed to support multi-tenancy, allowing different projects or tenants to have isolated networking environments within the same OpenStack deployment. This is crucial for cloud environments with multiple users or organizations.
- **Integration with OpenStack Services:**
- Neutron seamlessly integrates with other OpenStack services, particularly with the Compute service (Nova). This integration enables the creation and management of networks for virtual machines.

OpenStack Networking

- **What is Neutron**
- Neutron is an OpenStack project to provide "**Network connectivity as a Service (NaaS)**".
- The main process in OpenStack Networking is neutron-server, a Python daemon exposing the Networking API.
- **Functionality of Neutron:**
- Neutron enables users to create complex networking setups, define connectivity, and configure network policies in the cloud.
- It handles the creation and management of virtual networking components like networks, switches, subnets, and routers.
- Specifically works with devices managed by the OpenStack Compute service (nova).

OpenStack Networking

- OpenStack Networking integrates with various OpenStack components:

- **OpenStack Identity service (keystone)** is used for authentication and authorization of API requests.
- **OpenStack Compute service (nova)** is used to plug each virtual NIC on the VM into a particular network.
- **OpenStack Dashboard (horizon)** is used by administrators and project users to create and manage network services through a web-based graphical interface.

OpenStack Networking

- Why Neutron?
- **User Empowerment:**
- Neutron gives users the ability to easily create complex network setups using a simple API.
- *Example: You can use Neutron to build a multi-tier web application structure.*
- **Innovation with Plugins:**
- Neutron supports the use of additional features through plugins, whether they are open-source or closed-source.
- *Example: You can use plugins to implement advanced features like tunneling to overcome VLAN limits, ensure quality of service, or employ monitoring protocols.*
- **Building Advanced Network Services:**
- Neutron allows users to create and use advanced network services that seamlessly fit into their OpenStack environment.
- *Examples include services like Load Balancing, VPN, Firewall, IDS, and Data Center Interconnect.*

OpenStack Networking

- Why Neutron?
- **User-Friendly GUI:**
- Neutron integrates with the Horizon graphical user interface (GUI) for a visual and user-friendly experience. Users can create, delete, and manage L2 and L3 networks and subnets directly through the GUI.
- **API Extensibility:**
- Neutron provides an extensible API framework, allowing users to customize and extend its functionality.
- *Example: The "provider network" extension lets you map Neutron networks to specific VLANs in the physical data center, providing flexibility in network configuration.*

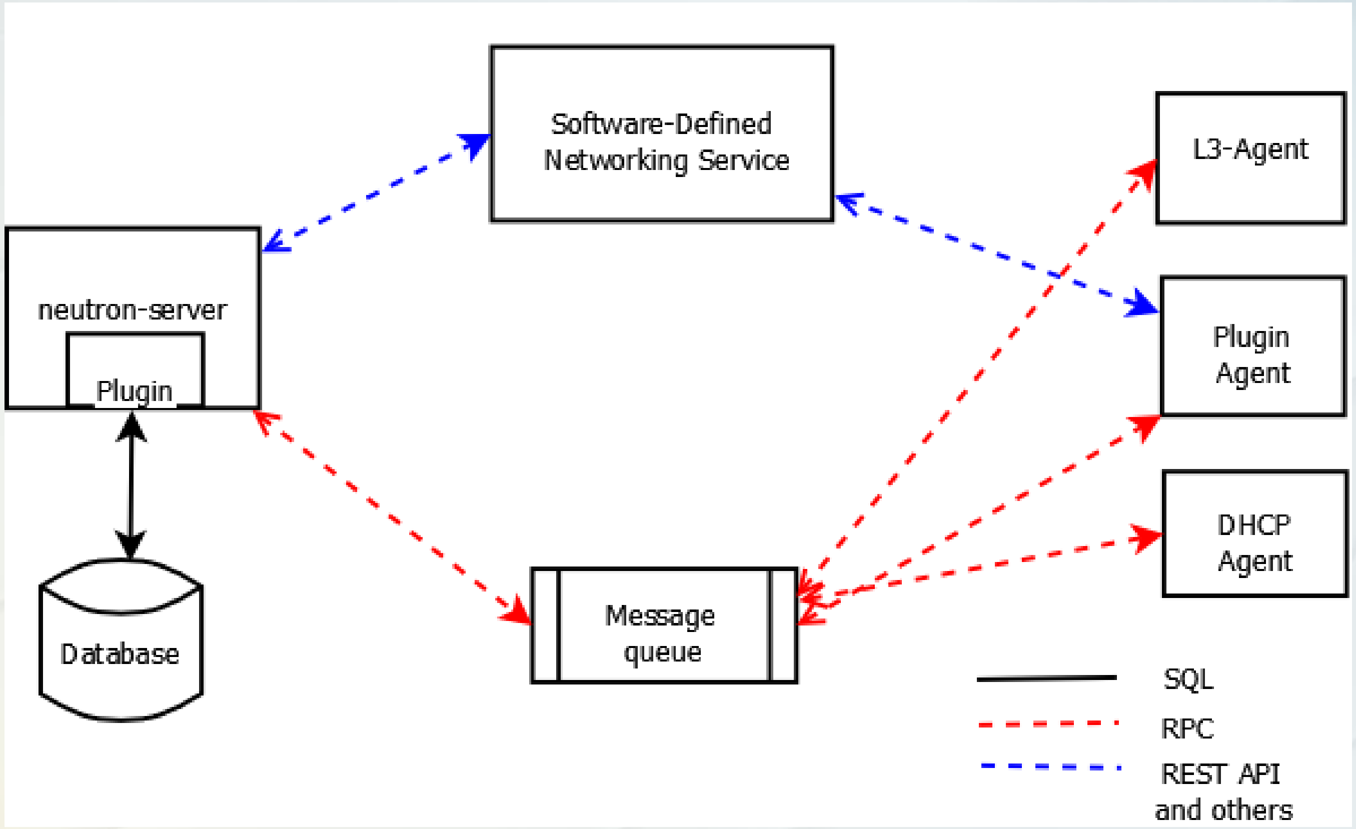
01

Architecture of Neutron

Neutron Plugins, Agents and Extensions

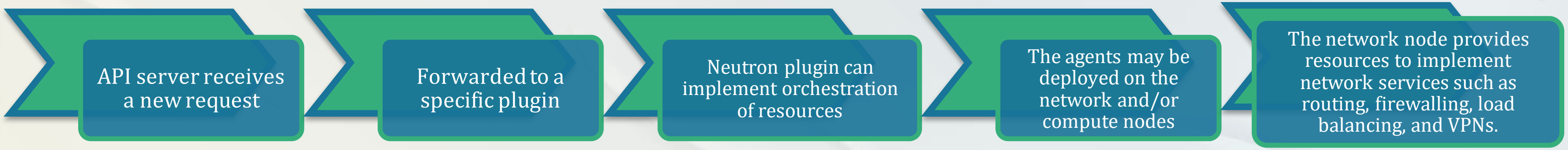
Module IV

Architecture of Neutron



Architecture of Neutron

- Neutron becomes a more and more effective and robust network project.
- It enables operators to build and manage a complete network topology with all the necessary elements including networks, subnets, routers, load balancers, firewalls, and ports.
- The Neutron project consists of an **API server**, which is responsible for receiving all networking service requests.
- Neutron has been designed to follow a **plugin-based architecture**


API server receives a new request
Forwarded to a specific plugin
Neutron plugin can implement orchestration of resources
The agents may be deployed on the network and/or compute nodes
The network node provides resources to implement network services such as routing, firewalling, load balancing, and VPNs.

Architecture of Neutron

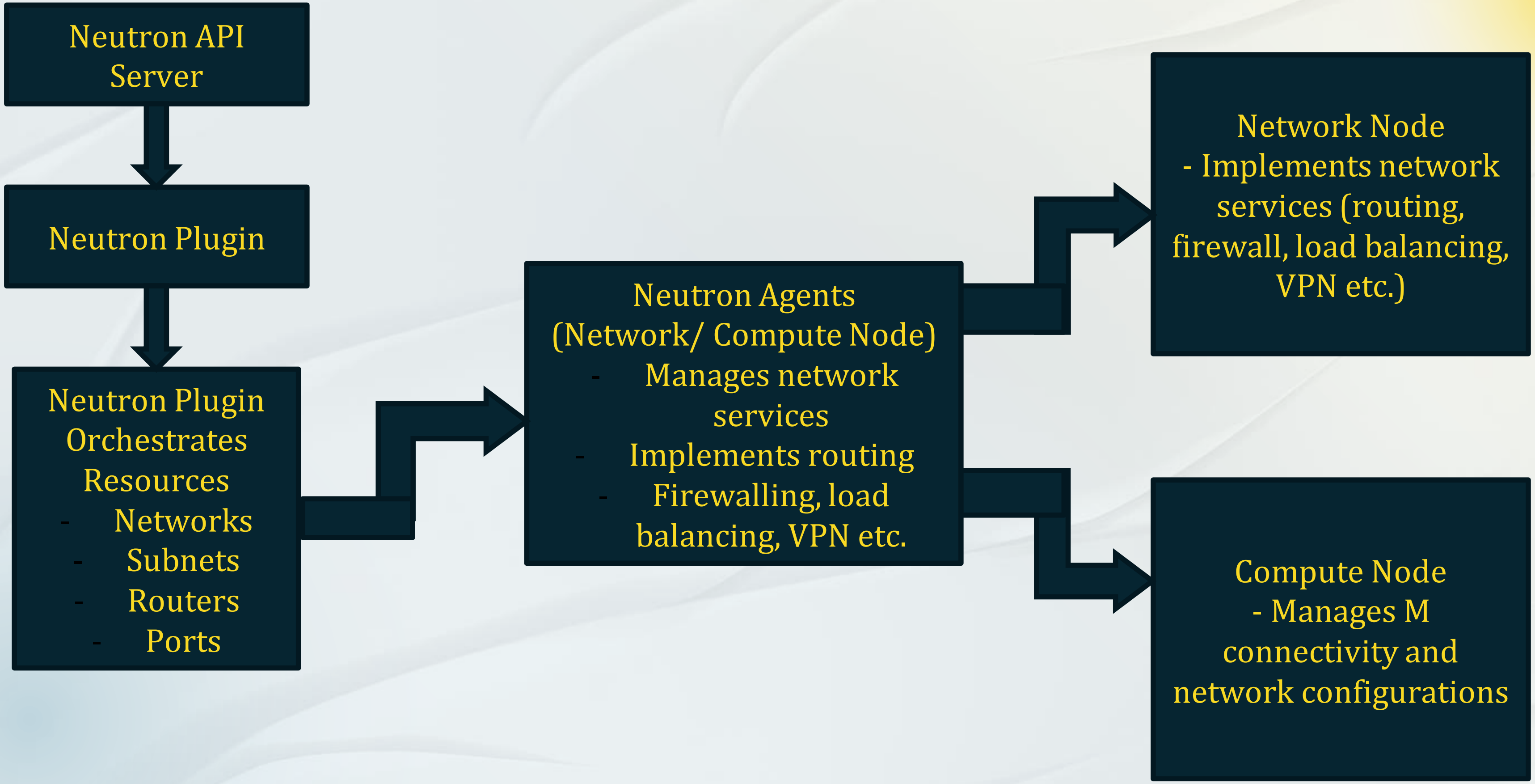
1. **API Request Received:**
    - The process begins when the Neutron API server receives a new request. This request could be from a user or another OpenStack service.
2. **Request Forwarded to Plugin:**
    - The Neutron API server forwards the request to a specific Neutron plugin. Neutron supports a pluggable architecture, allowing different plugins for various networking technologies.
3. **Plugin Orchestrates Resources:**
    - The Neutron plugin is responsible for orchestrating the necessary resources based on the received request.
    - This orchestration may involve creating, updating, or deleting network components such as networks, subnets, routers, and ports.

Architecture of Neutron

4. **Agent Deployment:**
   - Agents play a crucial role in Neutron's architecture. They may be deployed on network nodes, compute nodes, or both.
   - These agents facilitate communication and coordination between the Neutron server and the nodes in the OpenStack environment.
5. **Network Node Resources:**
   - On the network node, resources are provided to implement various network services.
   - Network services can include routing, firewalling, load balancing, and VPNs.
   - The network node acts as a central point for managing and controlling these network services.

Architecture of Neutron

6. **Compute Node Resources:**
   - Agents deployed on compute nodes handle the networking requirements of virtual machines (VMs) managed by the OpenStack Compute service (Nova).
   - These agents ensure that VMs have the necessary connectivity and network configurations.
7. **Collaboration Between Nodes:**
   - Collaboration between network nodes and compute nodes is essential for seamless networking within the OpenStack environment.
   - The network node and compute node agents communicate to ensure that the overall network setup aligns with the desired configuration.


Neutron API Server
Neutron Plugin
Neutron Plugin Orchestrates Resources
- Networks
- Subnets
- Routers
- Ports
Neutron Agents (Network/ Compute Node)
- Manages network services
- Implements routing
- Firewalling, load balancing, VPN etc.
Network Node
- Implements network services (routing, firewall, load balancing, VPN etc.)
Compute Node
- Manages M connectivity and network configurations

Neutron plugins

- *Networking for VM instances in OpenStack mirrors real-world networking principles.*
- *VM instances in OpenStack require at least Layer 2 (L2) network connectivity.*
- *VM instances may also need services like routing, firewall, and load balancing*.
- *Networking technologies and services can be implemented using a mix of both software and networking hardware.*
- *Neutron relies on plugins to support a wide range of networking technologies, providing flexibility and choice.*
- *Neutron plugins are pluggable Python classes invoked when responding to API requests.*

Neutron plugins

- Core and Service Plugins:
  - Neutron plugins are categorized into **Core** and **Service** plugins.
    - Core plugins handle fundamental L2 connectivity and IP address management.
    - Service plugins support additional services like Routing (L3), firewall, and load balancing.
  - Neutron plugins are like toolboxes that help manage networking features in OpenStack.
  - The Neutron plugins implement networking features by resource orchestration.

Plugins

The set of plugins originally included in the main Neutron distribution and supported by the Neutron community include:

- Open vSwitch Plugin
- Cisco UCS/Nexus Plugin
- Cisco ASR1000 Plugin
- Linux Bridge Plugin
- Modular Layer 2 Plugin
- Nicira Network Virtualization Platform (NVP) Plugin
- Ryu OpenFlow Controller Plugin
- NEC OpenFlow Plugin
- Big Switch Controller Plugin
- Cloudbase Hyper-V Plugin
- MidoNet Plugin
- Brocade Neutron Plugin Brocade Neutron Plugin
- PLUMgrid Plugin
- Mellanox Neutron Plugin Mellanox Neutron Plugin
- Embrane Neutron Plugin
- IBM SDN-VE Plugin
- CPLANE NETWORKS CPLANE NETWORKS
- Nuage Networks Plugin
- OpenContrail OpenContrail Plugin
- Lenovo Networking Lenovo Networking Plugin
- Avaya Neutron Plugin Avaya Neutron Plugin

Neutron plugins: Core Plugin

- Manages fundamental networking functionalities, such as Layer-2 connectivity and IP address management. It handles basic resources like networks, ports, and subnets.
- The core plugin focuses on making sure virtual machines and network parts can talk to each other (Layer-2 connectivity).
- Whenever someone wants to create a new virtual network or a new connection point (port), they talk to the core plugin through the API server.
- The core plugin implements the following Neutron resources:
- **Networks:** A Neutron network resource represents a Layer-2 domain. All virtual instances that need a network service must connect to a virtual network.
- **Ports:** A Neutron port represents a participating endpoint on the virtual network.
- **Subnet:** The Neutron subnet resource adds a Layer-3 addressing pool to a Neutron network. **The subnet** can have an associated DHCP server for serving IP address leases to virtual machines.

Neutron plugins : Core Plugin

Main Resources:

- **Networks:** Think of these as the virtual neighborhoods where devices live.
- **Ports:** These are like the virtual doors through which devices can communicate.
- **Subnets:** It adds a layer of addressing to the virtual neighborhood and can help devices get unique IP addresses.
- **API Interface**: The core plugin provides an interface (like a menu) that allows you to create and manage these virtual neighborhoods, doors, and addressing using simple commands
- The Neutron core's API is like a remote control; you use it to manage your virtual networks, doors, and addressing. You can create new neighborhoods, add new doors, and set up addressing pools for devices using this remote control.
- The Neutron core provides a RESTfull API interface to create and manage the mentioned resources.

Neutron plugins: SERVICE PLUGIN

- The service plugin is like a specialized toolkit for handling higher-level network services in OpenStack.
- Address higher-level networking services, including <u>routing, firewalling, load balancing, and VPNs.</u> Each service plugin focuses on a particular advanced networking feature..
- Example: Think of it like setting up road signs, traffic rules, security checks, and a receptionist for virtual networks.
- Different service plugins each handle a specific type of virtual network construction.
- Example: The L3 service plugin specializes in creating virtual routers that connect different virtual neighborhoods (networks).

Neutron plugins : SERVICE PLUGIN

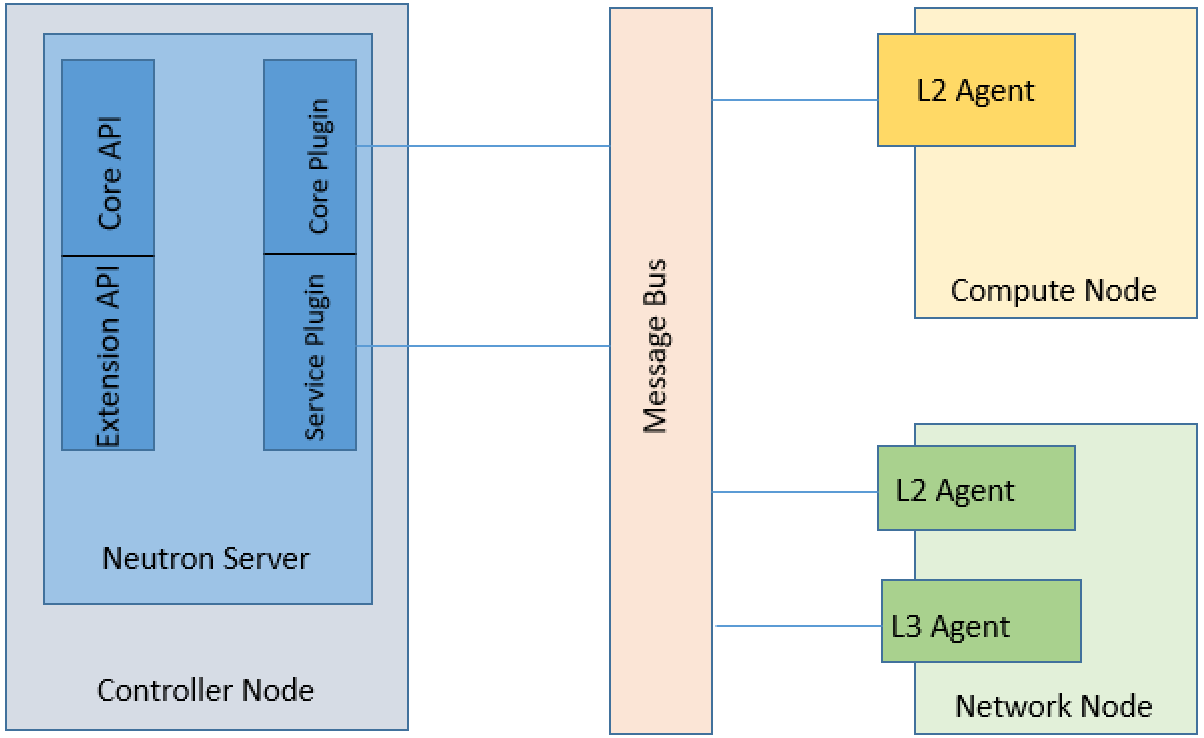
- **L3 Service Plugin Example:**
- Job of L3 Plugin: The L3 service plugin creates a virtual router that acts like a traffic cop, guiding data between two or more virtual neighborhoods (networks).
- Extra Resource Creation: It also creates something called a Floating IP, which works like a translator to help virtual machines communicate with the outside world.
- Think of the L3 service plugin as setting up a virtual highway between neighborhoods and providing a translator for communication beyond the neighborhood.

Agents

- The agents are deployed on the network and compute nodes.
- Agents are background processes responsible for performing specific networking tasks on network nodes or compute nodes.
- **Network Node Agent:** Handles networking services like routing, firewalling, load balancing, and VPNs at the network node level.
- **Compute Node Agent:** Manages the networking requirements of virtual machines (VMs) managed by the OpenStack Compute service (Nova) on compute nodes. Ensures VMs have the necessary connectivity and network configurations.
- Agents collaborate with the Neutron server to ensure the correct implementation and coordination of network-related tasks.
- They interact with the Neutron server using RPC calls over the message bus.

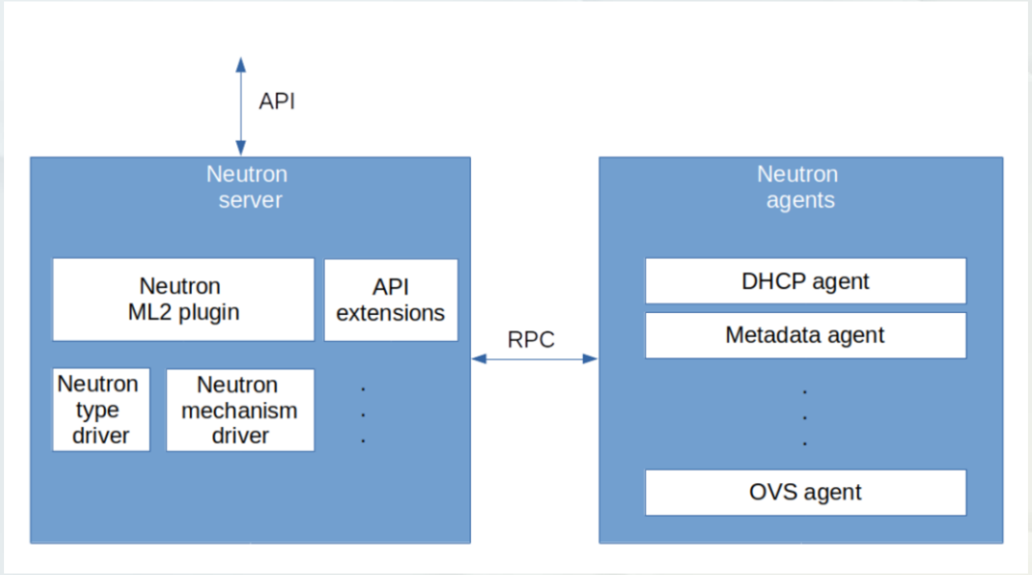
Agents

- Neutron provides different types to agents to implement virtual networking services such as Layer-2 connectivity, DHCP service, and routers.
- The following are some of the Neutron agents:
- The Neutron L2 agent resides on all the compute and network nodes. Its main function is to connect the virtual machines and network devices like virtual routers to the Layer-2 network. It interacts with the core plugin to receive network configuration for virtual machines.
- The DHCP agent is deployed on the network node and it implements DHCP service for a given subnet.
- The Neutron L3 agent implements a routing service and provides external access to a virtual machine by implementing NAT service.
- The VPN agent implement VPN service and is installed on the network nodes.


Core API
Extension API
Core Plugin
Service Plugin
Neutron Server
Controller Node
Message Bus
L2 Agent
Compute Node
L2 Agent
L3 Agent
Network Node

Neutron API extensions

- Extensions are provisions in the Neutron project for implementing advanced networking services. These extensions allow creating a new REST API and exposing Neutron resources. All advanced services are implemented as Neutron extensions.



Example: Creating a New Virtual Network

1. User Request:
   - A user, through the Horizon web interface or the command-line interface (CLI), requests the creation of a new virtual network.
2. API Request Received:
   - Neutron's API server receives the request.
3. Core Plugin Activation:
   - The API server delegates the task to the core plugin, which is responsible for handling fundamental networking tasks, including creating networks.
4. Plugin Orchestrates Resources:
   - The core plugin orchestrates the necessary resources based on the request. In this case, it involves creating a new network.
5. RPC Message Creation:
   - The core plugin creates an RPC (Remote Procedure Call) message that contains instructions for the agents to implement the changes.

Example: Creating a New Virtual Network

6. Message Sent to Appropriate Agents:
   - The RPC message is sent to the relevant agents. For network-related tasks, this may involve both the Network Node Agent and Compute Node Agents.
7. Agent Actions:
   - Network Node Agent:
     - Receives the RPC message.
     - Takes actions related to network services such as routing, firewalling, and load balancing at the network node level.
   - Compute Node Agent:
     - Receives the RPC message.
     - Ensures that VMs on the compute node have the necessary connectivity and configurations.

Example: Creating a New Virtual Network

8. Agent-Plugin Collaboration:
    - The agents collaborate with the core plugin to implement the requested changes.
    - The Network Node Agent configures the network-related services on the network node.
    - The Compute Node Agent ensures that the VMs have the required network configurations.
9. Network Configuration:
    - The agents make the necessary changes to the local networking components, such as configuring the Open vSwitch (OVS) on the network node and ensuring VM connectivity on the compute node.
10. Once the changes are implemented, the core plugin, agents, and Neutron API server collaborate to confirm the successful creation of the virtual network. The user is notified through the interface that the new virtual network is ready for use.

02

Implementing Virtual Networks

VLAN and Tunnel Based

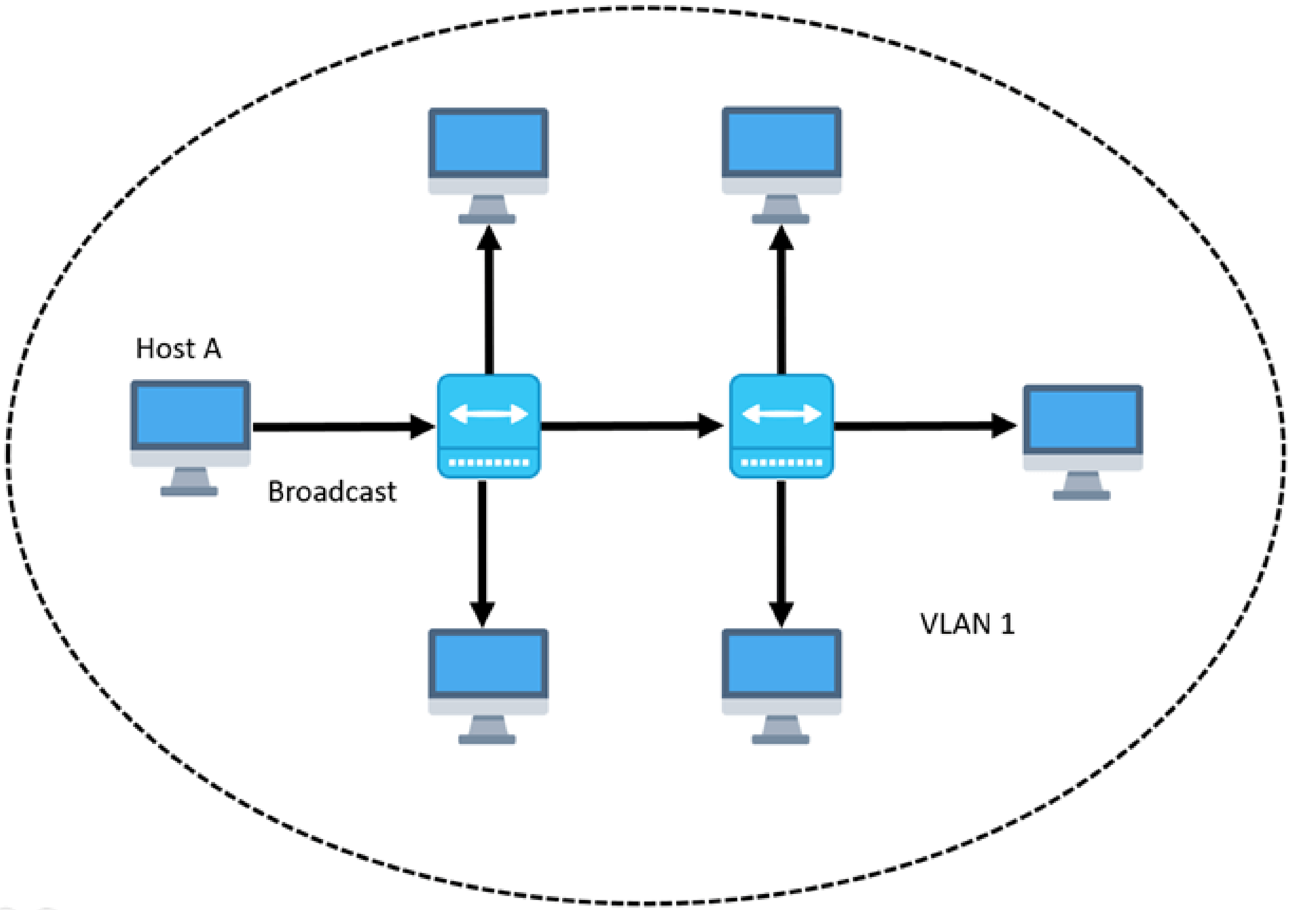
Module IV

Implementing virtual networks

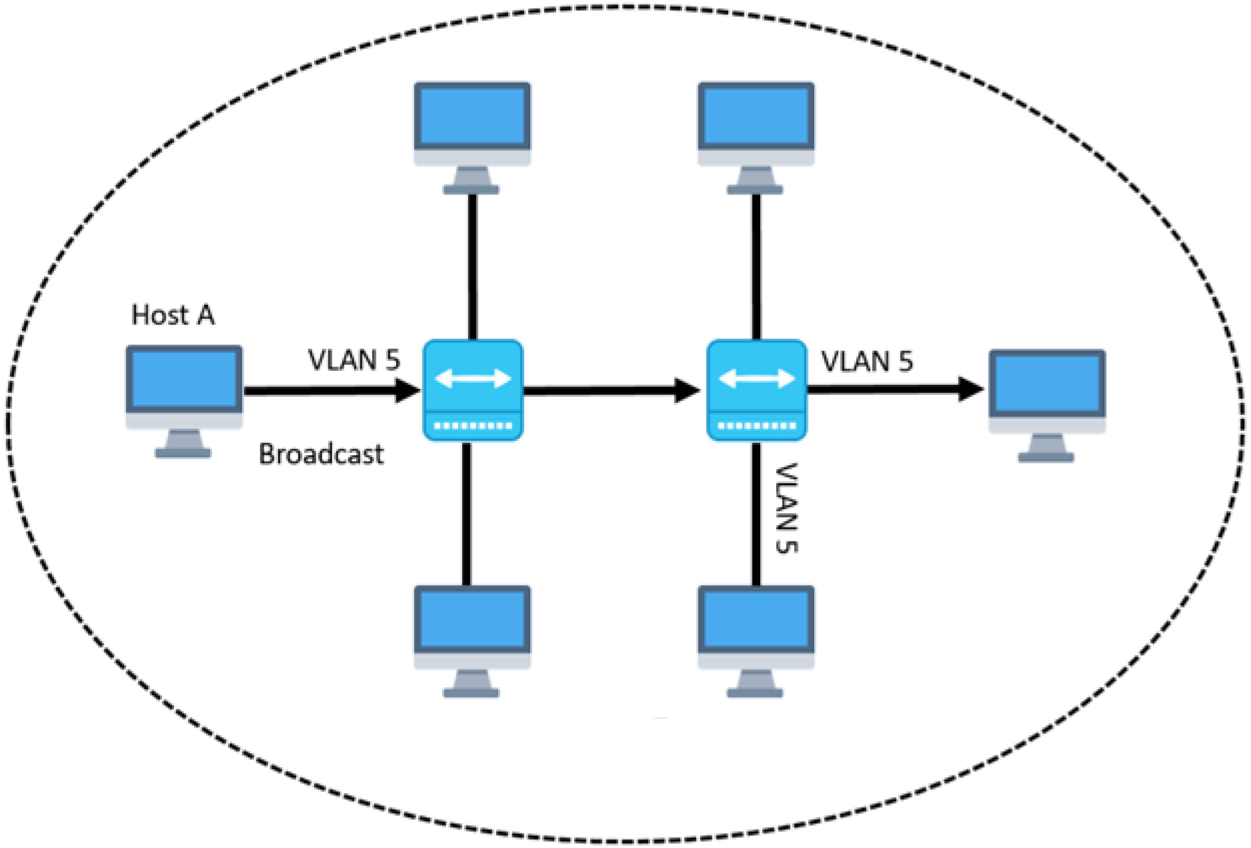
- Neutron core plugins handle the creation of virtual networks and ports.
- A network in Neutron is a single Layer-2 broadcast domain.
- Each virtual network created is associated with a separate Layer-2 domain; this helps in keeping the traffic within a virtual network isolated.
- The Neutron virtual networks can be created in multiple ways but broadly they can be categorized into VLAN-based and tunnel-based networks.
- VLAN-based Networks
- Tunnel Based Networks

VLAN Based Networks

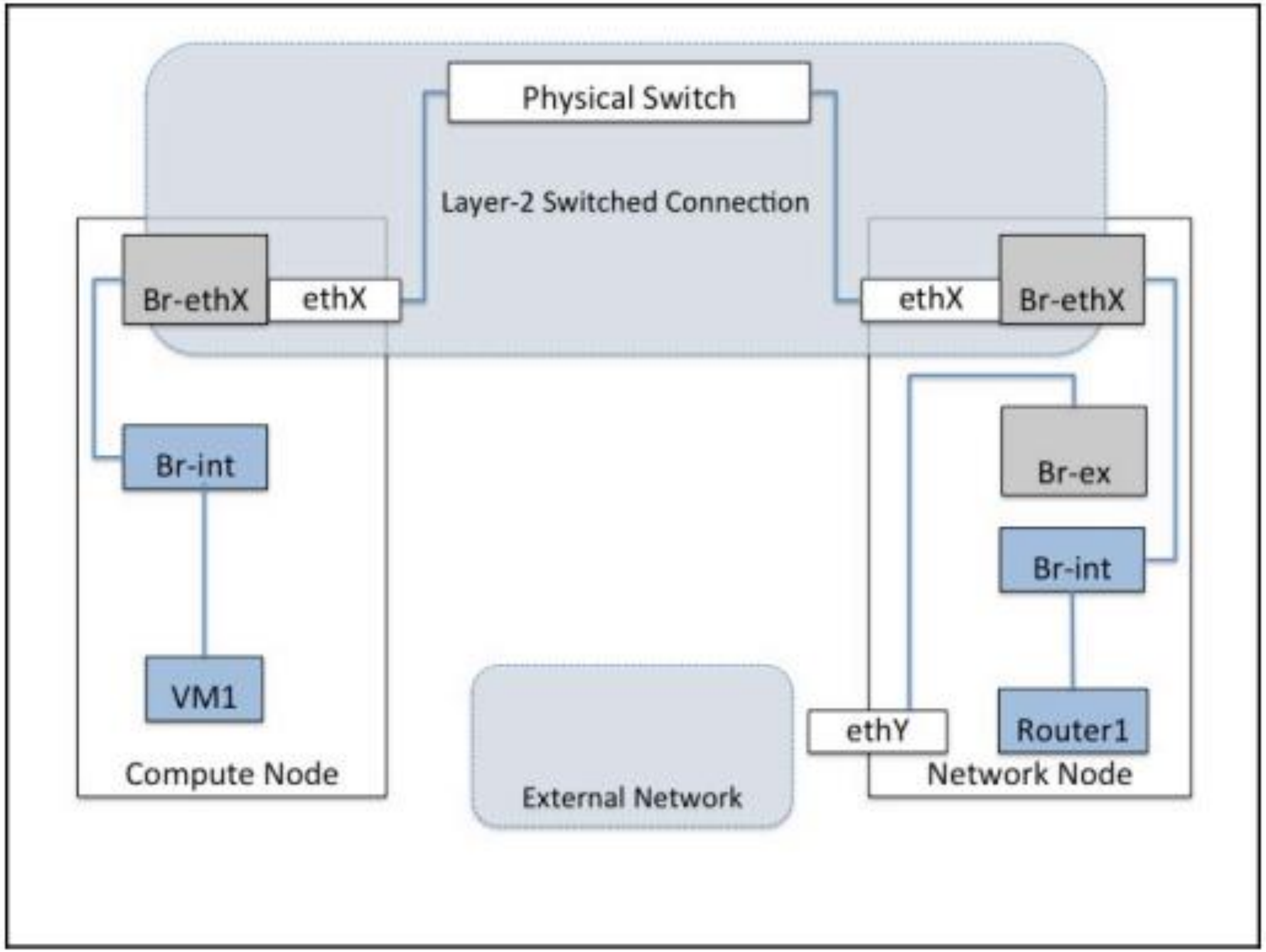
- **What is VLAN?**
- VLAN is a custom network which is created from one or more local area networks.
- It enables a group of devices available in multiple networks to be combined into one logical network.
- The result becomes a virtual LAN that is administered like a physical LAN.
- The full form of VLAN is defined as Virtual Local Area Network.
- Without VLANs, a broadcast sent from a host can easily reach all network devices.
- Each and every device will process broadcast received frames.
- It can increase the CPU overhead on each device and reduce the overall network security.
- In case if you place interfaces on both switches into separate VLAN, a broadcast from host A can reach only devices available inside the same VLAN.
- Hosts of VLANs will not even be aware that the communication took place. This is shown in the below picture: The below topology depicts a network having all hosts inside the same virtual LAN:



network having all hosts inside the same VLAN



Host A can reach only devices available inside the same VLAN

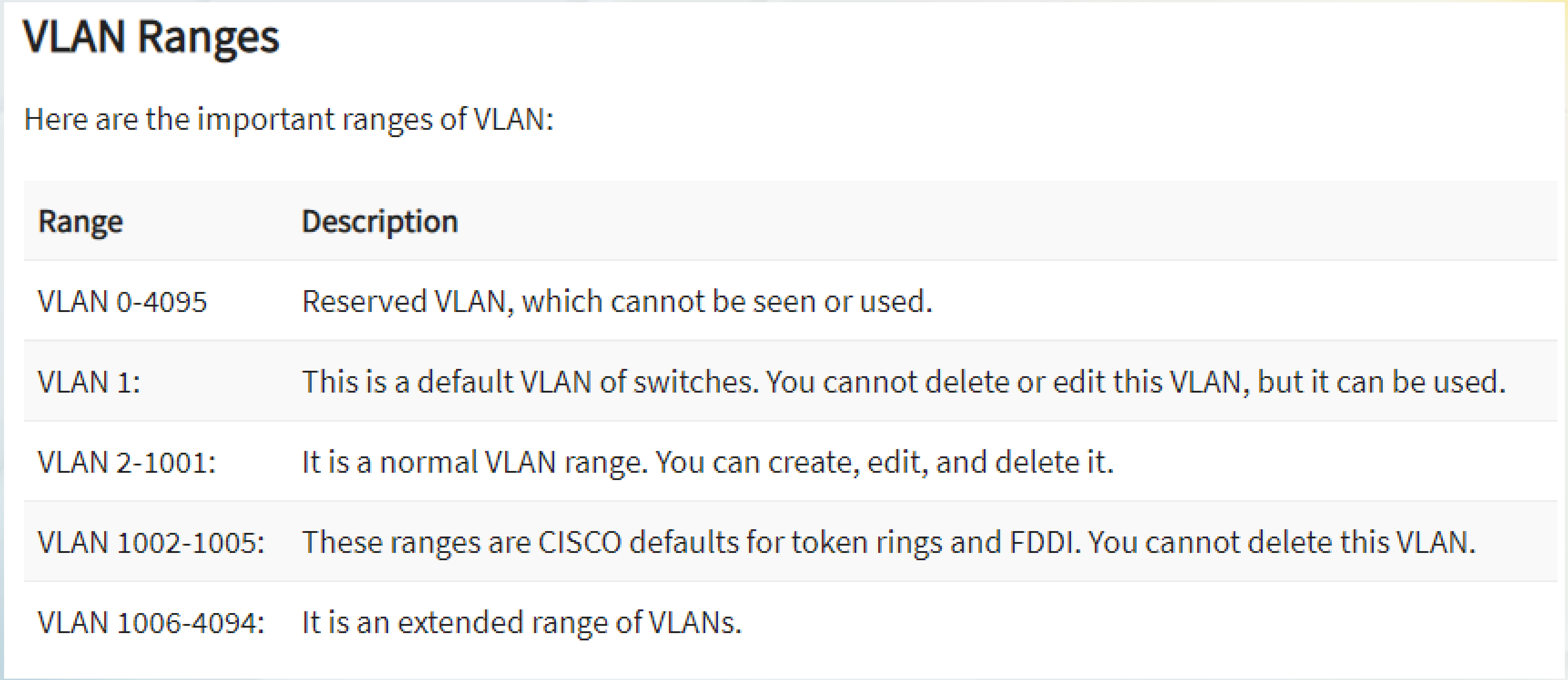

Physical Switch
Layer-2 Switched Connection
Br-ethX
ethX
ethX
Br-ethX
Br-int
Br-ex
Br-int
VM1
Router1
ethY
Compute Node
External Network
Network Node

How VLAN works ?

- VLAN is a logical network created within a physical network.
- It allows devices from different physical networks to be grouped together as if they are on the same network.
- VLANs are used to segment a physical network into multiple logical networks.
- Each VLAN represents a separate broadcast domain, isolating traffic within its boundaries.
- VLANs are identified by a numerical tag, ranging from 1 to 4094.
- This VLAN ID is associated with frames to indicate the VLAN to which they belong.
- Switch ports are assigned to specific VLANs.
- Devices connected to these ports are considered part of the assigned VLAN.

How VLAN works

- Broadcast traffic is contained within the VLAN.
- Devices within the same VLAN receive and process broadcasts, while devices in other VLANs remain unaffected.
- VLANs enable communication between switches.
- Ports on each switch are assigned the same VLAN for communication, typically using trunk links.
- EXAMPLE
- Consider a network with three VLANs: VLAN 1, VLAN 2, and VLAN 3.
- Devices in VLAN 1 communicate within their VLAN, and the same applies to devices in VLAN 2 and VLAN 3.
- Inter-VLAN communication can be facilitated through routing or layer-3 switches.

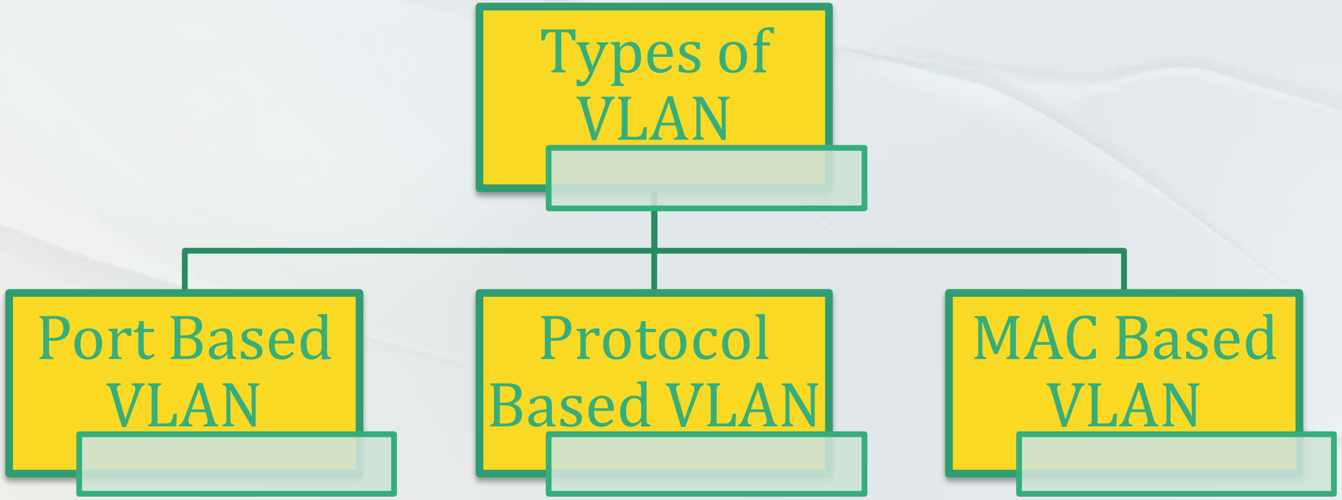


VLAN Ranges

Here are the important ranges of VLAN:

Range	Description
VLAN 0-4095	Reserved VLAN, which cannot be seen or used.
VLAN 1:	This is a default VLAN of switches. You cannot delete or edit this VLAN, but it can be used.
VLAN 2-1001:	It is a normal VLAN range. You can create, edit, and delete it.
VLAN 1002-1005:	These ranges are CISCO defaults for token rings and FDDI. You cannot delete this VLAN.
VLAN 1006-4094:	It is an extended range of VLANs.

Types of VLANs



How VLAN works

- **Port-Based VLAN**
- Port-based VLANs groups virtual local area network by port. In this type of virtual LAN, a switch port can be configured manually to a member of VLAN.
- **Protocol Based VLAN**
- This type of VLAN processes traffic based on a protocol that can be used to define filtering criteria for tags, which are untagged packets.
- **MAC Based VLAN**
- MAC Based VLAN allows incoming untagged packets to be assigned virtual LAN and, thereby, classify traffic depending on the packet source address. You define a Mac address to VLAN mapping by configuring mapping the entry in MAC to the VLAN table.

Difference between LAN and VLAN

LAN	VLAN
• LAN can be defined as a group of computer and peripheral devices which are connected in a limited area.	• A VLAN can be defined as a custom network which is created from one or more local area networks.
• The latency of LAN is high.	• The latency of VLAN is less.
• The cost of LAN is high.	• The cost of a VLAN is less.
• In LAN, the network packet is advertised to each and every device.	• In VLAN, the network packet is sent to only a specific broadcast domain.
• It uses a ring, and FDDI (Fiber Distributed Data Interface) is a protocol.	• It uses ISP and VTP as a protocol.

Application/Purpose of VLAN

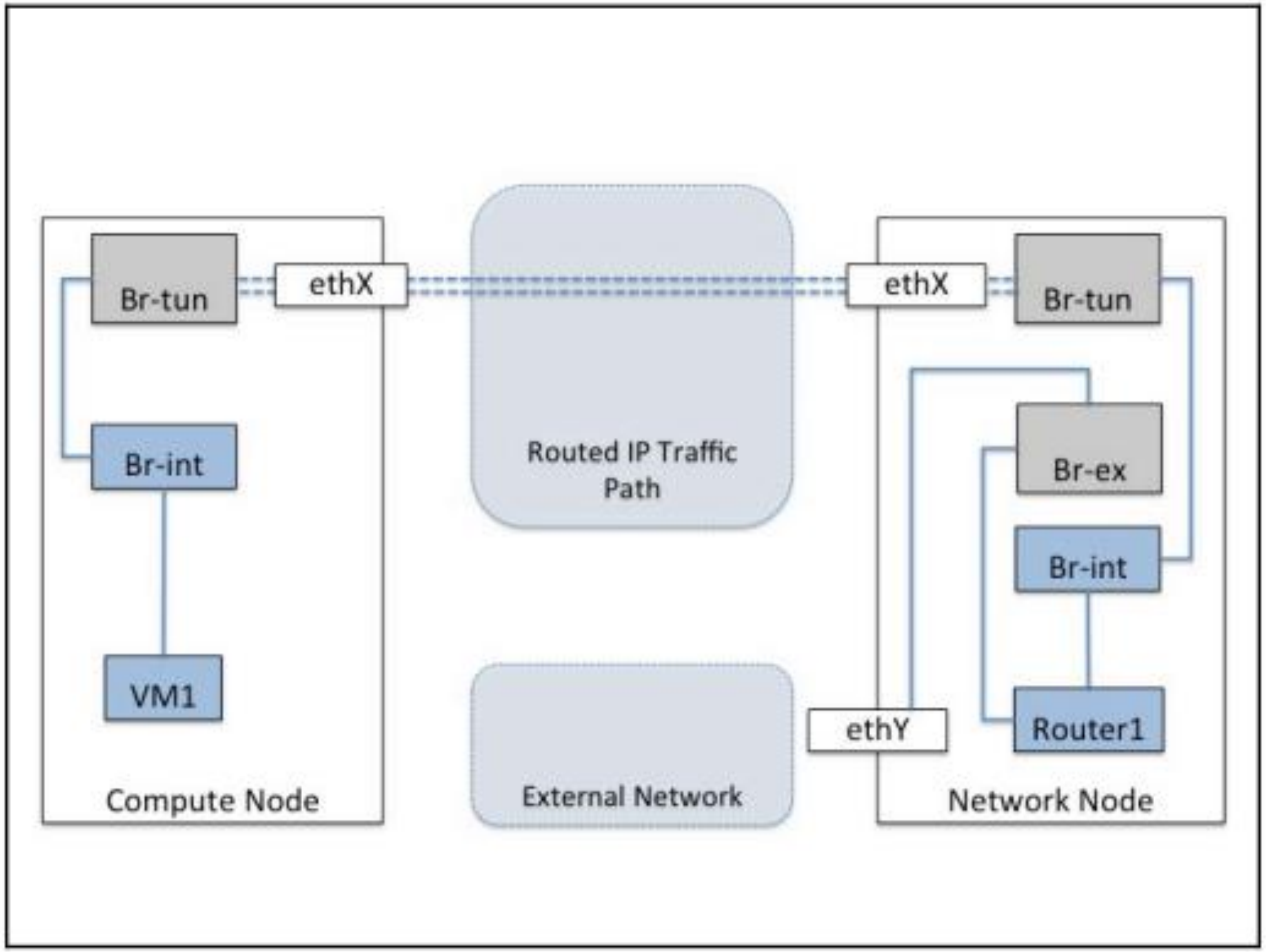
- VLAN is used when you have 200+ devices on your LAN.
- It is helpful when you have a lot of traffic on a LAN.
- VLAN is ideal when a group of users need more security or being slow down by many broadcasts.
- It is used when users are not on one broadcast domain.
- Make a single switch into multiple switches.

Tunnel Based Networks

- A **tunnel-based VLAN** involves creating a virtual network that spans across different physical networks or locations.
- It's like creating a secret passageway (**tunnel**) that allows communication between devices as if they were on the same local network, even though they might be physically distant.
- Tunnel-based VLANs use a technique called **encapsulation, packet encapsulation (PE)**. It's like putting the original data (packets) inside another protective layer for secure transportation.
- These VLANs create **overlay** networks. Imagine a **transparent layer over your regular network that connects devices seamlessly, regardless of their physical location.**
- The encapsulated data travels through existing networks, like the internet or private connections, but it remains protected and isolated as it passes through various routers and switches.

Tunnel Based Networks

- Tunnel-based VLANs can operate at both Layer 2 (Data Link Layer) and Layer 3 (Network Layer) of the OSI model, depending on the specific tunneling protocol used.
- EXAMPLE: If you have offices in different cities and want employees to feel like they're on the same local network, you might use tunnel-based VLANs. Devices in one city communicate as if they are in the same room, thanks to the virtual tunnel connecting them.
- Tunnel-based VLANs offer flexibility for creating virtual networks without being constrained by physical network boundaries. This is particularly useful in cloud environments or for connecting geographically dispersed locations.
- Examples of tunneling protocols include VXLAN, GRE, and Geneve. Each has its specific way of creating and managing these virtual tunnels.


Br-tun
ethX
Routed IP Traffic Path
ethX
Br-tun
Br-int
Br-ex
Br-int
VM1
External Network
ethY
Router1
Compute Node
Network Node

Process of sending L2 packets in Tunnel Based Networks

1. Initial Setup:
    - There are two nodes involved - a compute node and a network node.
    - A virtual machine (VM1) resides on the compute node.
    - Virtual routers are established on both the compute node and the network node.
2. VM Communication:
    - VM1 on the compute node wants to communicate with another port on a virtual network.
3. Packet Transmission:
    - VM1 sends out a packet to the virtual router's interface (router port) within the same virtual network.
4. Encapsulation at Compute Node:
    - The virtual switch on the compute node encapsulates the Layer-2 (L2) packet sent by VM1 inside an IP packet.
    - This encapsulation allows the L2 packet to be transported over an IP Fabric to the network node.

Process of sending L2 packets in Tunnel Based Networks

5. Transmission Over IP Fabric:
   - The encapsulated packet is sent over the IP Fabric, a network that connects the compute node and the network node.
6. Decapsulation at Network Node:
   - Upon reaching the network node, the virtual switch on the destination node removes the outer IP packet, revealing the original Layer-2 packet sent by VM1.
7. Delivery to Destination Router:
   - The recovered Layer-2 packet is then delivered to the destination virtual router port on the network node.
8. Emulating Layer-2 Network:
   - Despite the fact that the compute node and the network node are not part of a single Layer-2 network, the use of tunnels allows them to emulate a Layer-2 network for connected virtual machines.

Tunneling Process:

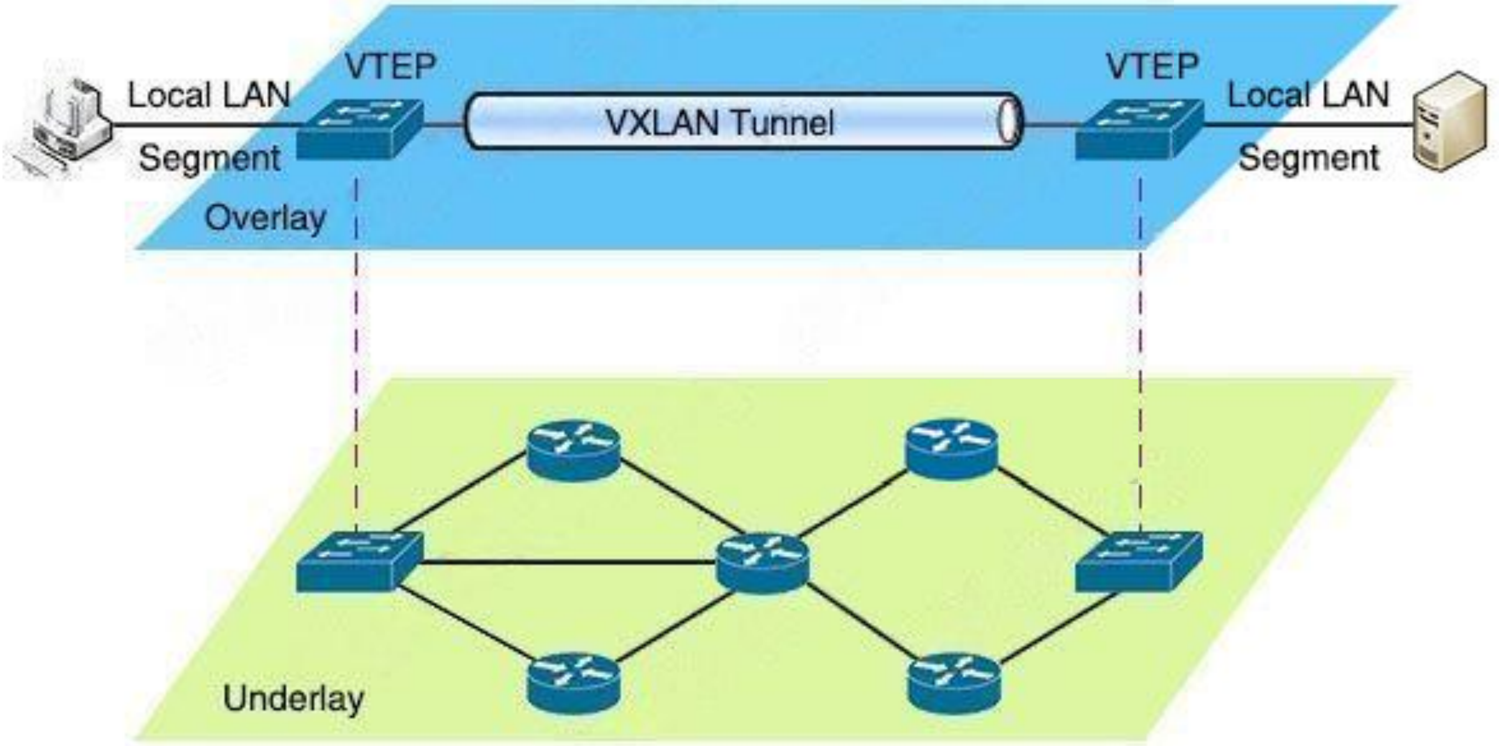
The encapsulation and decapsulation process is a form of tunneling, where data is wrapped in an additional layer for secure transport.

Emulating Layer-2 Network:

The described process enables the virtual machines to communicate as if they were part of the same Layer-2 network, even though they are on different nodes.

Components of a tunnel based network:

- **Underlay network**: An underlay network is the IP based network connectivity (IP Fabric) between the compute and network nodes.
- **Overlay network**: An overlay network is a virtual Layer-2 domain created by encapsulating the packets from the virtual machines in to IP packets. The overlay network uses the underlay network as a means of transporting packets, while for the underlay network, the overlay packets are similar to any other data packet flowing through the IP network.
- Protocols to implement a tunnel-based virtual network
- **Virtual Extensible Local Area Network (VXLAN)**
- **GRE-based tunnels.**
- Geneve
- The tunnel-based networks provide huge segmentation size; VXLAN, for instance, can create 16 million separate networks.

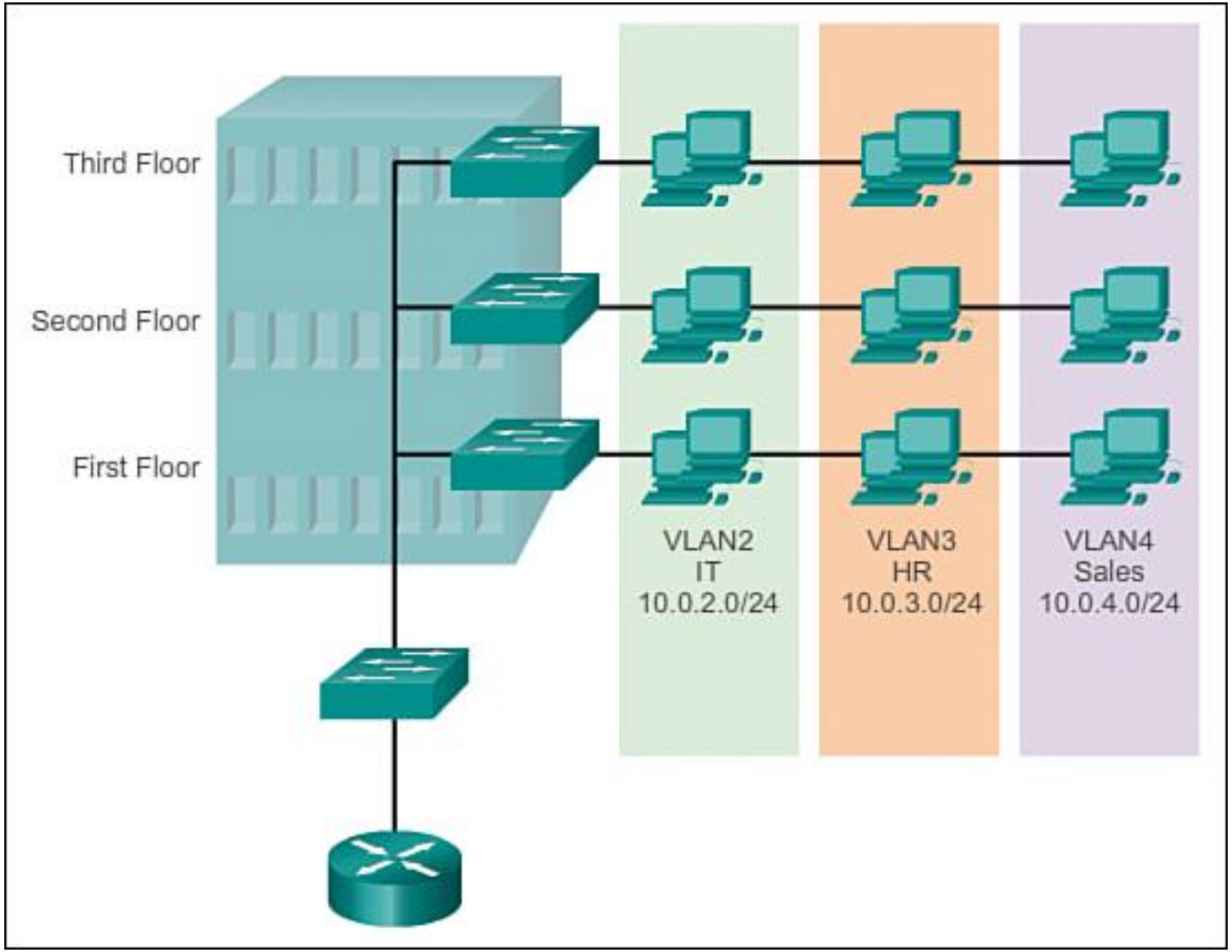

VTEP
VTEP
Local LAN
Segment
VXLAN Tunnel
Local LAN
Segment
Overlay
Underlay

Tunnel based network:

- **Virtual Extensible Local Area Network (VXLAN)**
- An overlay technology that helps address scalability issues with VLANs.
- VXLAN encapsulates layer 2 Ethernet frames inside layer 4 UDP packets that can be forwarded or routed between hosts.
- This means that a virtual network can be transparently extended across a large network without any changes to the end hosts.
- In the case of OpenStack Networking, however, a VXLAN mesh network is commonly constructed only between nodes that exist in the same cloud.
- **GRE-based tunnels.**
- A GRE network is similar to a VXLAN network in that traffic from one instance to another is encapsulated and sent over a layer 3 network.
- A unique segmentation ID is used to differentiate traffic from other GRE networks.
- Rather than use UDP as the transport mechanism, GRE uses IP protocol 47.
- For various reasons, the use of GRE for encapsulating tenant network traffic has fallen out of favor now that VXLAN is supported by both Open vSwitch and Linux Bridge network agents.

Tunnel based network:

- Network segmentation divides the network into different zones or segments.
- This can be done with physically different Local Area Networks (cables) or the segmentation may be implemented by the network switches and firewalls – known as Virtual LANs or VLANs.
- Network segmentation with virtual local area networks (VLANs) creates a collection of isolated networks within the data center. Each network is a separate broadcast domain
- VLAN technology was created to reduce broadcast traffic that affects network speed.
- VLAN allows you to divide the network into segments, and in the event of a broadcast storm or other problems not the entire network, but only part of it may collapse.


Third Floor
Second Floor
First Floor
VLAN2
IT
10.0.2.0/24
VLAN3
HR
10.0.3.0/24
VLAN4
Sales
10.0.4.0/24

What would be the most suitable network segmentation mechanism?

- The common argument against using VLAN-based network segmentation is the limited number of segmentations that can be created.
- The VLAN ID is part of the Layer-2 header of the packet and its size is well defined.
- This puts a limit of 4048 distinct VLAN segmentations and only as many virtual networks can be created with VLANs, while the **number of tunnel-based networks that can be created is much higher, in the order of millions.**
- Another important feature of the tunneled networks is their **ability to span across multiple L2 domains in the underlay**.
- This means it is possible to create tunneled networks with ports residing in different datacenters, thus scaling the cloud deployment to multiple locations.

What would be the most suitable network segmentation mechanism?

- The VLAN-based network is **much better performing** compared to the tunnel-based ones, as there is no overhead of encapsulating and de- encapsulating of packets.
- Additionally, due to the encapsulation of packets in a tunnel- based network, each packet flowing through the overlay network carries a tunnel header and an IP header.
- This is an overhead on the IP Fabric and has an impact on the maximum size of packet a VM can send through the network.
- The encapsulation overhead can be tweaked either by adjusting the maximum transfer unit (MTU) size of the underlay network to accommodate the increased packet size, or by reducing the MTU size of the virtual network.

03

VirtualSwitches, ML2 Plugin, Neutron Subnet

Module IV

Virtual Switches

- A virtual switch is a software-based networking component that operates at the data link layer (Layer 2) of the OSI (Open Systems Interconnection) model.
- It is used in virtualized environments, such as virtual machines (VMs) or containers, to enable communication between virtual entities within a host or across multiple hosts.
- Virtual switches play a crucial role in the networking infrastructure of virtualized environments, providing connectivity and managing the flow of network traffic.

Virtual Switches

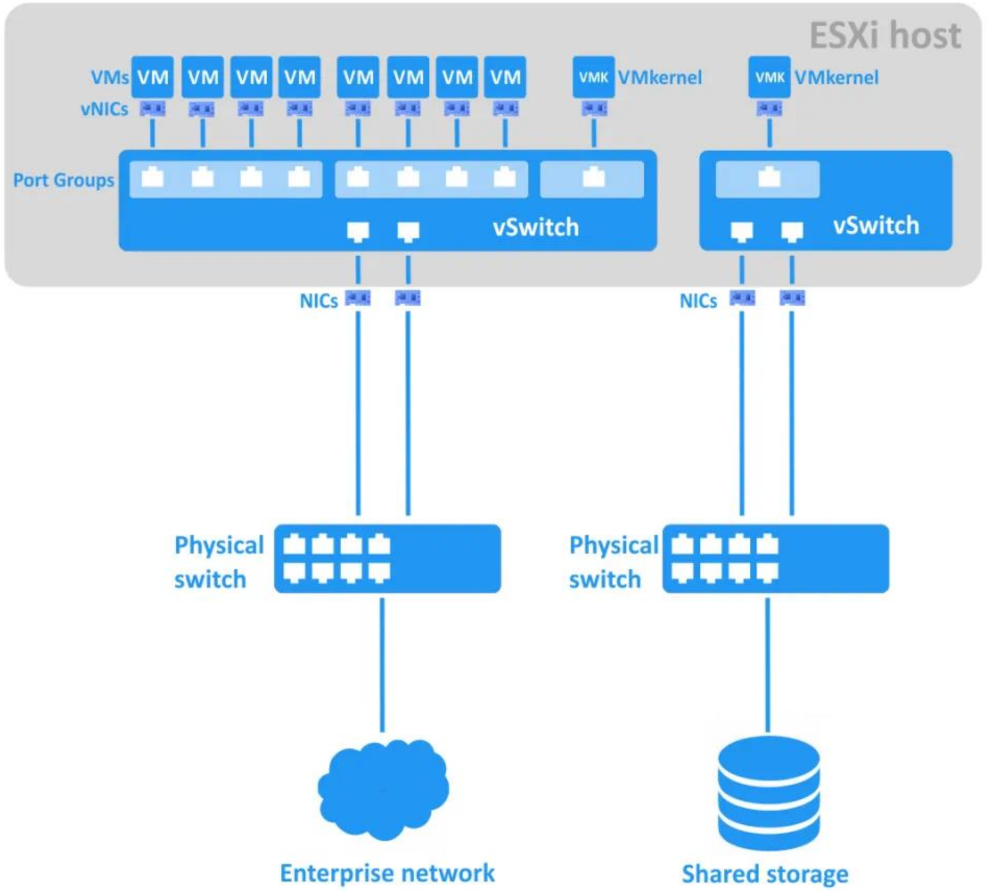
- Connectivity: Virtual switches connect virtual machines, containers, or other virtualized network entities within a host or across multiple hosts. They enable these entities to communicate with each other as if they were connected to a physical switch.
- Isolation: Virtual switches help isolate the network traffic of different virtual entities. This isolation ensures that the communication within one virtual environment does not interfere with or impact other virtual environments running on the same host or across different hosts.
- Port Groups: Virtual switches typically have port groups that define sets of ports with similar characteristics or configurations. For example, a port group might be associated with a specific VLAN (Virtual Local Area Network).

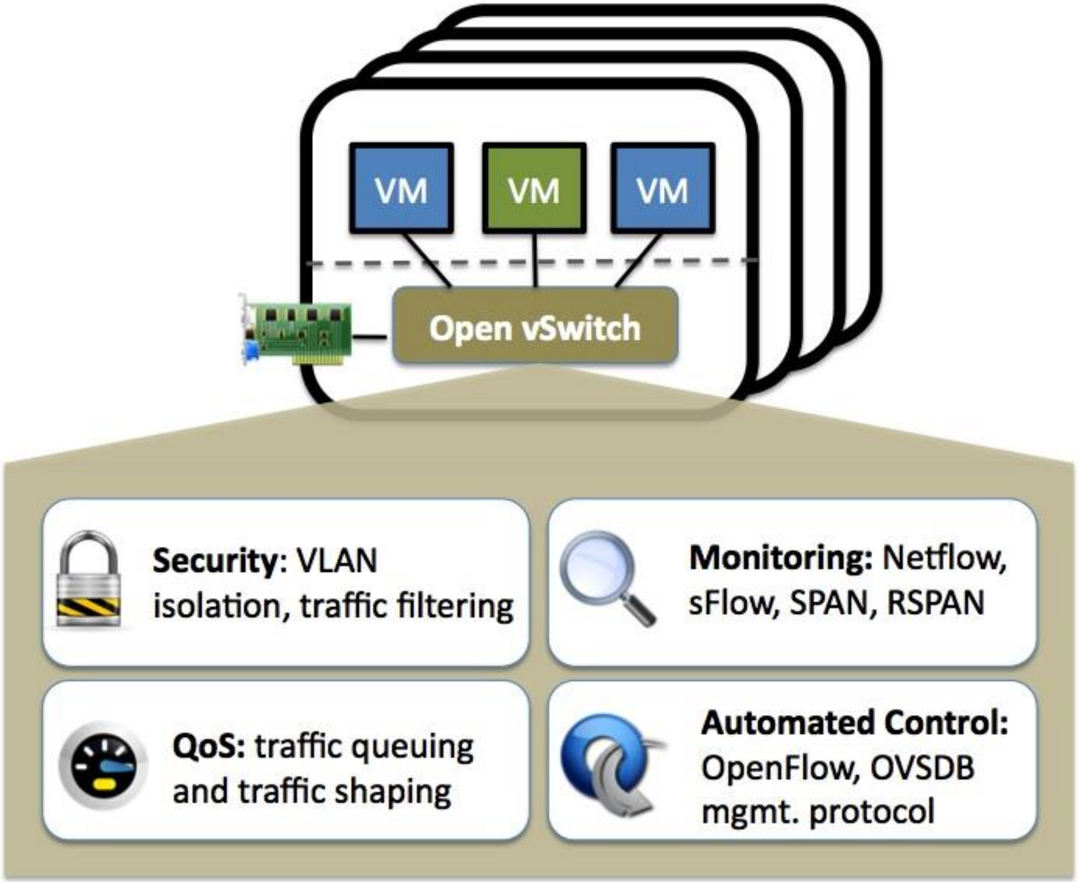
Virtual Switches

- VLAN Support: Many virtual switches support Virtual LANs (VLANs), allowing for the segmentation of network traffic into different logical networks. This segmentation helps in organizing and securing network traffic.
- Traffic Filtering and Forwarding: Virtual switches can perform traffic filtering and forwarding functions. They inspect incoming network packets, make decisions based on defined rules, and forward the packets to the appropriate destination.
- Integration with Hypervisor: Virtual switches are often integrated with hypervisors (such as VMware vSphere, Microsoft Hyper-V, or KVM) and are managed through the hypervisor's management interface.

Virtual Switches

- Software-Defined Networking (SDN) Integration: In some cases, virtual switches are part of broader software-defined networking solutions, allowing for more programmable and flexible network configurations.
- Popular examples of virtual switches include the **VMware vSphere Standard Switch (vSwitch)** and **Distributed Switch (vDS), Microsoft Hyper-V Virtual Switch**, and **Open vSwitch**.


ESXi host
VMs
VM
VM
VM
VM
VM
VM
VM
VM
VMK
VMkernel
VMK
VMkernel
vNICs
Port Groups
vSwitch
vSwitch
NICs
NICs
Physical switch
Physical switch
Enterprise network
Shared storage

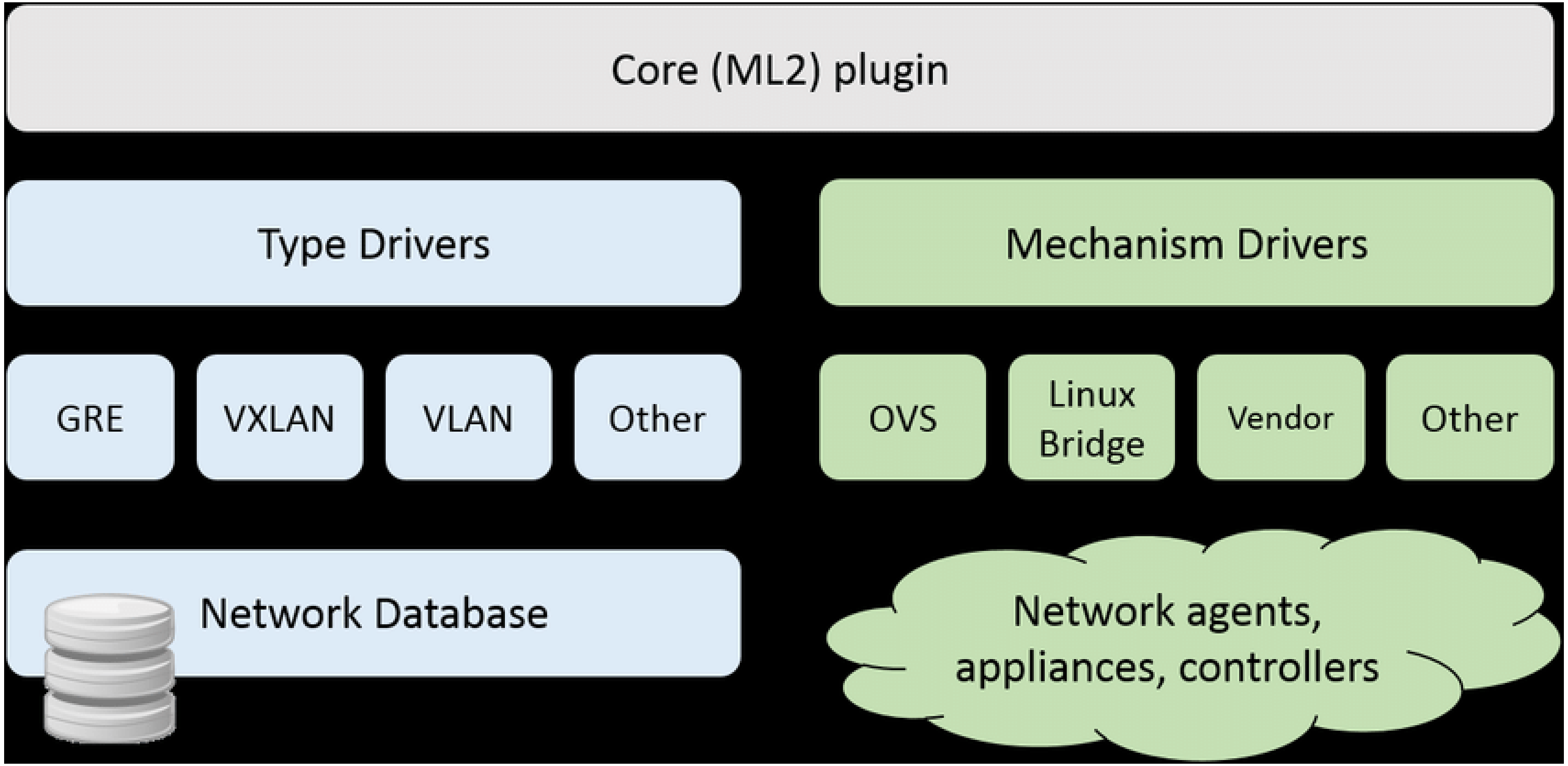

VM
VM
VM
Open vSwitch
Security: VLAN isolation, traffic filtering
Monitoring: Netflow, sFlow, SPAN, RSPAN
QoS: traffic queuing and traffic shaping
Automated Control: OpenFlow, OVSDB mgmt. protocol

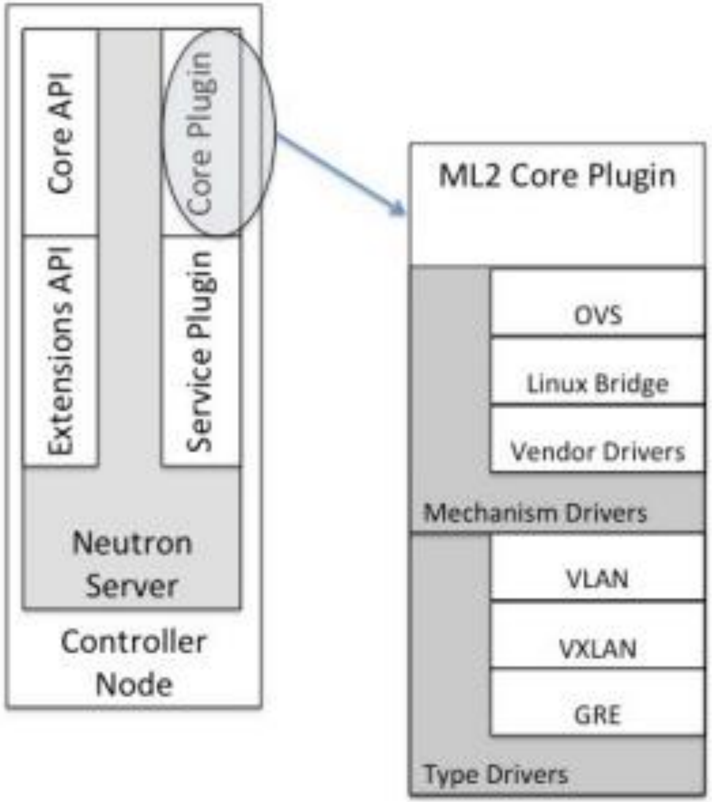
The ML2 Plugin

- In OpenStack, ML2 refers to the Modular Layer 2 plugin, which is a framework for managing layer 2 (L2) networking in OpenStack.
- ML2 allows the coexistence of multiple plugins, enabling administrators to choose the networking technology that best suits their needs.
- ML2 supports various networking technologies, such as VLANs, VXLAN, and GRE.
- The ML2 core plugins can include:
- MechanismDriver: This is a core component of the ML2 plugin responsible for interacting with the underlying networking infrastructure. It abstracts the networking details and allows ML2 to work with different networking technologies.
- TypeDriver: This component is responsible for managing the type of network segmentation, such as VLANs, VXLANs, or GRE. It provides a layer of abstraction, allowing OpenStack to support different types of network segmentation without requiring changes to the core code.

The ML2 Plugin

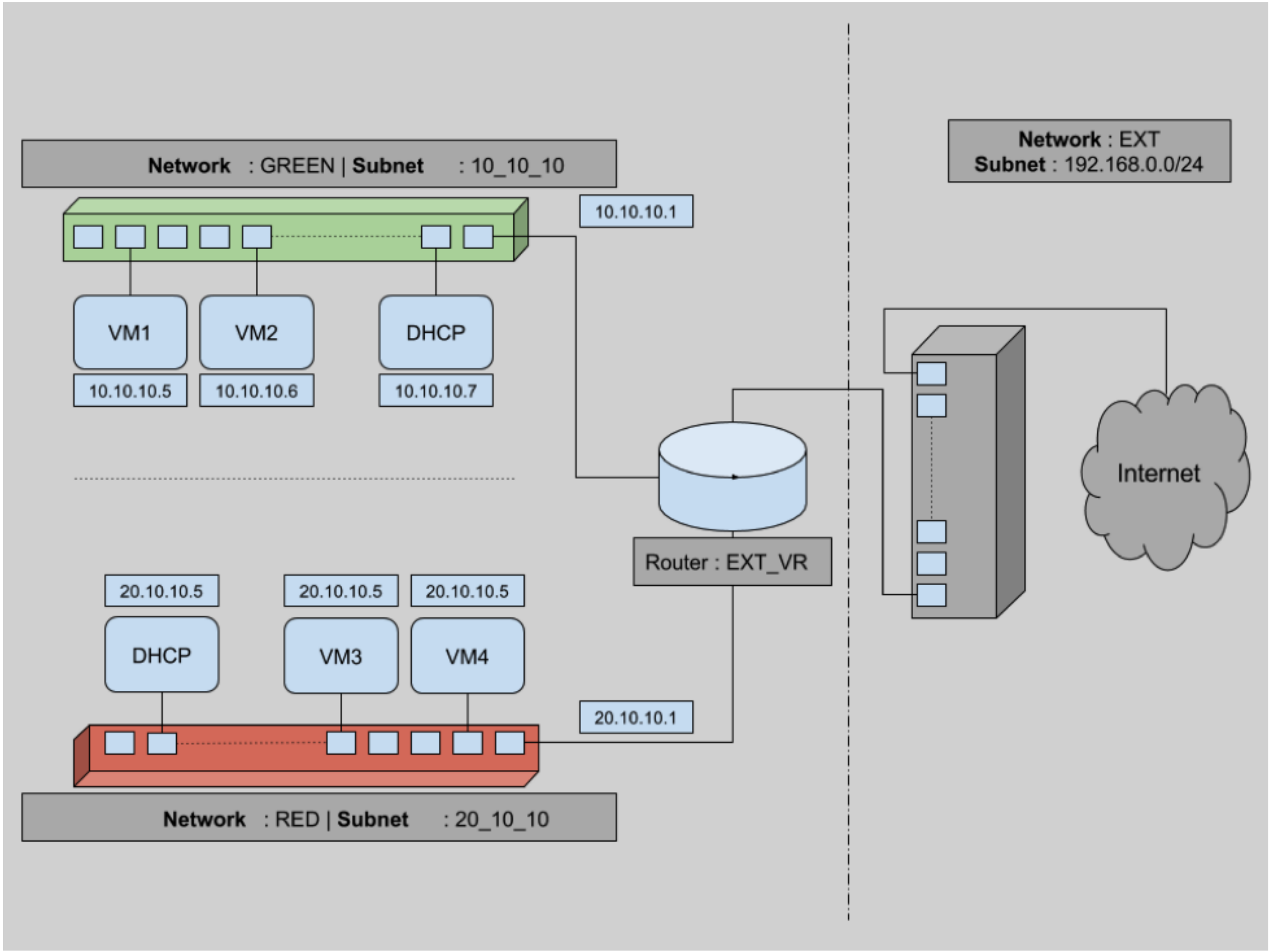
- Types of ML2 core plugins include:
- TypeDriver for VLAN (VLAN TypeDriver): Manages VLAN-based network segmentation. Each VLAN represents a separate logical network.
- TypeDriver for VXLAN (VXLAN TypeDriver): Manages VXLAN-based network segmentation. VXLAN is a tunneling protocol commonly used in virtualized environments.
- TypeDriver for GRE (GRE TypeDriver): Manages GRE-based network segmentation. GRE is another tunneling protocol that allows the creation of private, point-to-point connections.


Core (ML2) plugin
Type Drivers
Mechanism Drivers
GRE
VXLAN
VLAN
Other
OVS
Linux Bridge
Vendor
Other
Network Database
Network agents, appliances, controllers


Core API
Extensions API
Core Plugin
Service Plugin
Neutron Server
Controller Node
ML2 Core Plugin
OVS
Linux Bridge
Vendor Drivers
Mechanism Drivers
VLAN
VXLAN
GRE
Type Drivers

Neutron subnets

- The subnets define a pool of IP addresses associated to a network.
- The definition of subnets in Neutron contains the IP pool in the form of Classless Inter-Domain Routing (CIDR).
- Optionally, the gateway of the subnet can be adjusted by default as the first IP address in the pool.
- It should be noted that Neutron allows the association of multiple subnets to a single network.



Network : GREEN | **Subnet** : 10_10_10
10.10.10.1
VM1
VM2
DHCP
10.10.10.5
10.10.10.6
10.10.10.7
Network : EXT
Subnet : 192.168.0.0/24
Internet
Router : EXT_VR
20.10.10.5
20.10.10.5
20.10.10.5
DHCP
VM3
VM4
20.10.10.1
Network : RED | **Subnet** : 20_10_10

04

Connecting Virtual Networks

Module IV

1. Connecting Virtual Networks with routers

- Connecting Virtual Machines (VMs) in a Virtual Network:
  - A virtual network allows multiple **virtual machines to connect** and communicate within the same network.
- Crossing Layer-2 Network Boundaries:
  - To enable communication across different virtual networks (**crossing Layer-2 network boundaries**), **virtual routers** come into play.
- Functionality of Virtual Routers:
  - Virtual routers facilitate the connection of multiple networks.
  - Tenants (users or administrators) add the subnet associated with a virtual network to the virtual router.
- Configuration of Virtual Router Ports:
  - Adding a subnet to the virtual router creates a port on the virtual network.
  - This port is assigned the IP address of the gateway for the subnet.

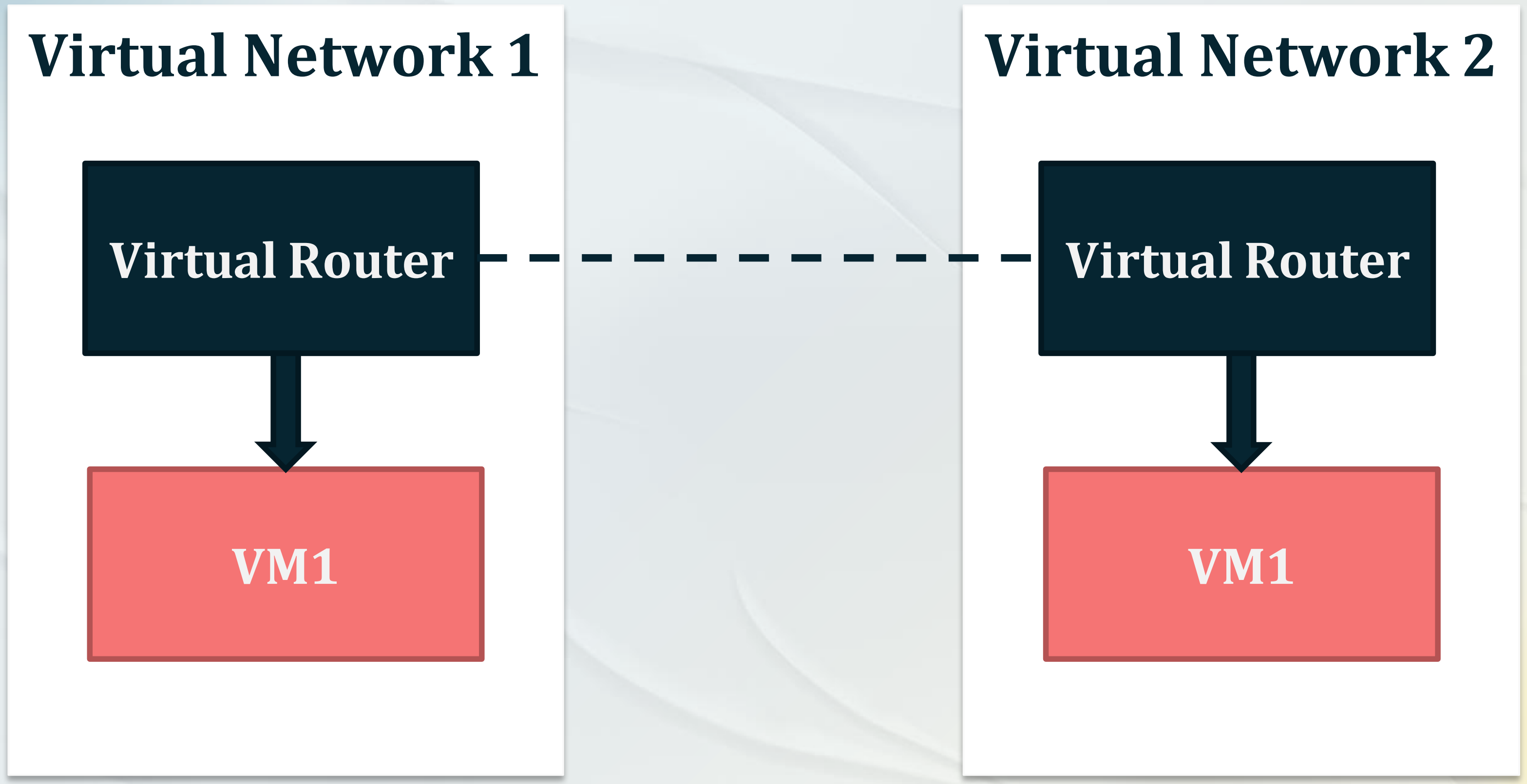
1. Connecting Virtual Networks with routers

- DHCP Server and IP Address Assignment:
  - When virtual machines are assigned IP addresses by a DHCP server on the network, the DHCP offer includes the IP address of the network gateway.
- IP Packet Forwarding:
  - The virtual router plays a crucial role in forwarding IP packets between the connected networks, allowing communication across different subnets.

1. Connecting Virtual Networks with routers

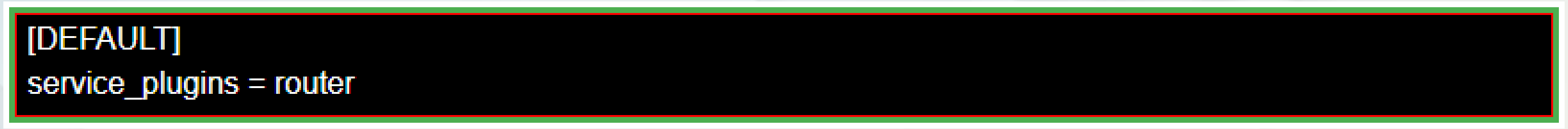
- Implementation Using Network Namespaces:
  - Think of a network namespace as a way to create isolated copies of the networking environment on a Linux system.
  - In OpenStack's Neutron networking component, there's a default tool called the Layer 3 (L3) plugin. This plugin helps in creating virtual routers.
  - When you make a virtual router using Neutron's L3 plugin, it uses **Linux network namespaces** in the background.
  - Each virtual router gets its own separate network namespace. It's like giving each router its own private space to do its networking stuff.
  - Inside this separate space (the network namespace), a virtual router has its own set of interfaces (connections), routes (paths for data), and firewall rules (security settings).
  - In the context of Neutron, this setup ensures that each virtual router works independently. Each router is like its own mini-world, thanks to the magic of Linux network namespaces.

1. Connecting Virtual Networks with routers



2. Configuring the routing service

- To provide the routing service, **Neutron server must be configured with the router service plugin.**
- To do this, update the service plugin list in Neutron configuration file /etc/neutron/neutron.conf:

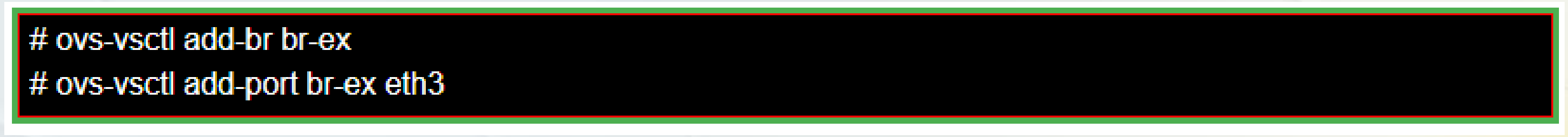


```
[DEFAULT]
service_plugins = router
```

- The router plugin implements virtual router instances <u>using Linux namespaces.</u>
- This is done using the **L3 agent** deployed on the network node.
- The **router plugin and L3 agent** communicate over the message bus.
- The L3 agent configuration file is present at **/etc/neutron/l3_agent.ini**:

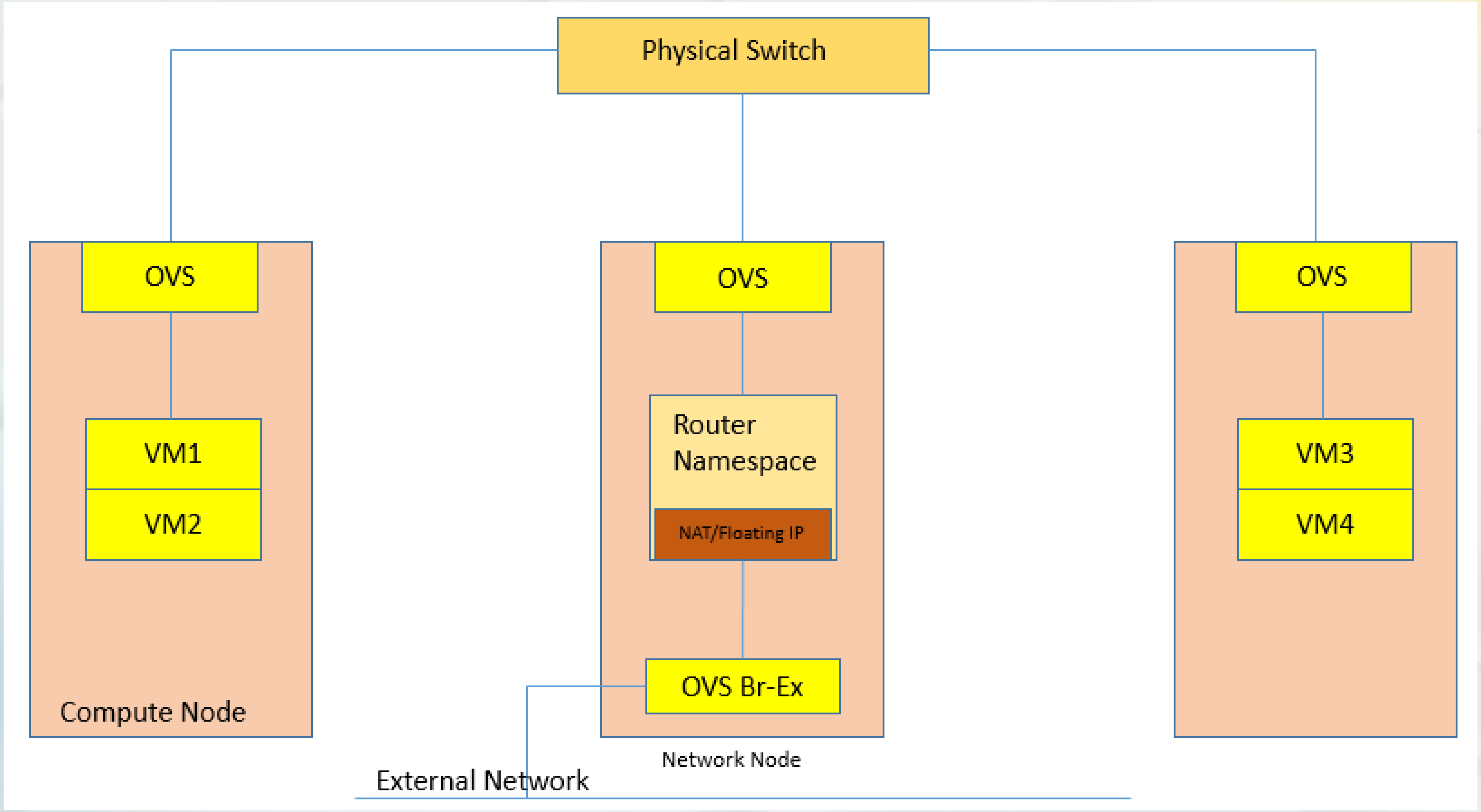
2. Configuring the routing service

- The **router plugin** also provides external access using **NAT and floating IP configuration.**
- NAT and floating IP requires a configured network node with the external network bridge.
- Assuming the third NIC on the network node is used for providing external access, use the following OVS commands to create the external access bridge:



```
# ovs-vsctl add-br br-ex
# ovs-vsctl add-port br-ex eth3
```

2. Configuring Routing Service:



3. Connecting networks using a virtual router

- To create a virtual router use the router-create command as follows:

```
$ neutron router-create router1
```

- Next add the subnets to the router using the router-interface-add command:

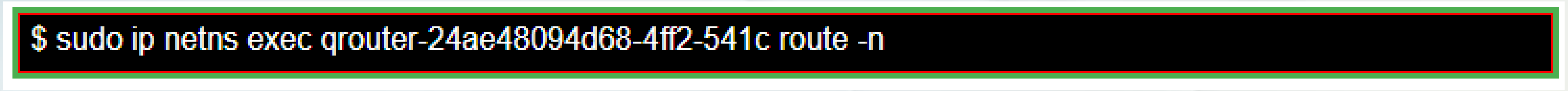
```
$ neutron router-interface-add subnet1
```

- Once the router interfaces are added, you can check the namespace created by the L3 agent on the network node. To do this, use the ip netns commands as follows:

```
$ ip netns list
qdhcp-22bda338-303a-4aae-9a55
qrouter-24ae48094d68-4ff2-541c
```

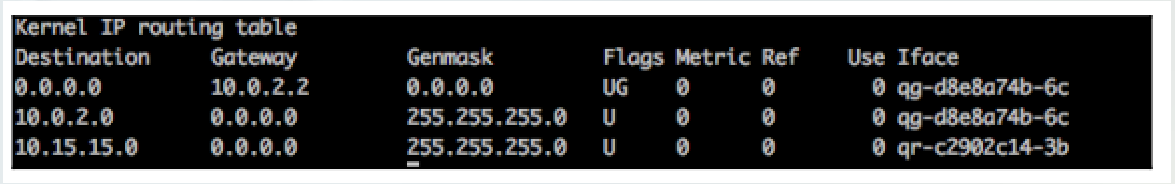
3. Connecting networks using a virtual router

- 65ef2787-541c-4ff2-8b69-24ae48094d68 is the router id.
- Next you can check the routing table in the virtual router, as follows, by selecting the router namespace:



```
$ sudo ip netns exec qrouter-24ae48094d68-4ff2-541c route -n
```

- Output



```
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         10.0.2.2        0.0.0.0         UG    0      0        0 qg-d8e8a74b-6c
10.0.2.0        0.0.0.0         255.255.255.0   U     0      0        0 qg-d8e8a74b-6c
10.15.15.0      0.0.0.0         255.255.255.0   U     0      0        0 qr-c2902c14-3b
```

4. Connecting to the external world

- Different ways to connect
- **Approach One: Direct Connection to Internet-Accessible Network**
  - The simplest method is to directly connect the virtual machine to a network with Internet access.
  - For instance, linking virtual machines to the management network can grant them Internet access.
  - Admins can create a provider network, making the management network available to OpenStack users.

4.Connecting to the external world

- **Approach Two: Using a Router for Internet Access**
  - The alternative is employing a router to enable Internet access.
  - Earlier discussions highlighted connecting multiple networks. Now, the focus is on accessing the Internet using a virtual router.
  - While theoretically possible to send packets to the outside world via the virtual router, there's a challenge.
  - The IP addresses used for creating subnets are non-routable from the Internet.
  - Consequently, even if packets are sent out, responses cannot reach back due to the non-routable IP addresses.

4. Connecting to the external world

- Solving the Internet Access Challenge: Two Options
- Using Routable Addresses:
  - Getting a large pool of routable IP addresses can be costly.
  - Assigning routable addresses directly to virtual networks.
- Using Network Address Translation (NAT):
  - NAT is used to translate IP addresses within packet headers during transit.
  - NAT involves modifying the IP address information in packet headers during transit.
  - The idea of NAT is to allow multiple devices to access the Internet through a single public address.
  - To achieve this, the translation of a private IP address to a public IP address is required.
  - Network Address Translation (NAT) is a process in which one or more local IP address is translated into one or more Global IP address and vice versa in order to provide Internet access to the local hosts.

4. Connecting to the external world

- Source Network Address Translation (SNAT):
  - Purpose: Allows multiple devices in a local network to share a single public IP address.
  - How it Works:
    - A router connected to an external network uses its external IP for SNAT.
    - Internal devices' source addresses are translated to the router's external IP.
    - Responses from the Internet are directed to the router, which then translates and forwards them to the internal device.

4. Connecting to the external world

- Destination Network Address Translation (DNAT):
  - Purpose: Redirects incoming packets destined for a specific external IP to an internal device.
  - How it Works:
    - A router with DNAT configuration receives packets addressed to its external IP.
    - The router translates the destination address to the internal IP and forwards the packet to the corresponding internal device.
    - Responses from the internal device are sent to the Internet with the source address translated back to the router's external IP.

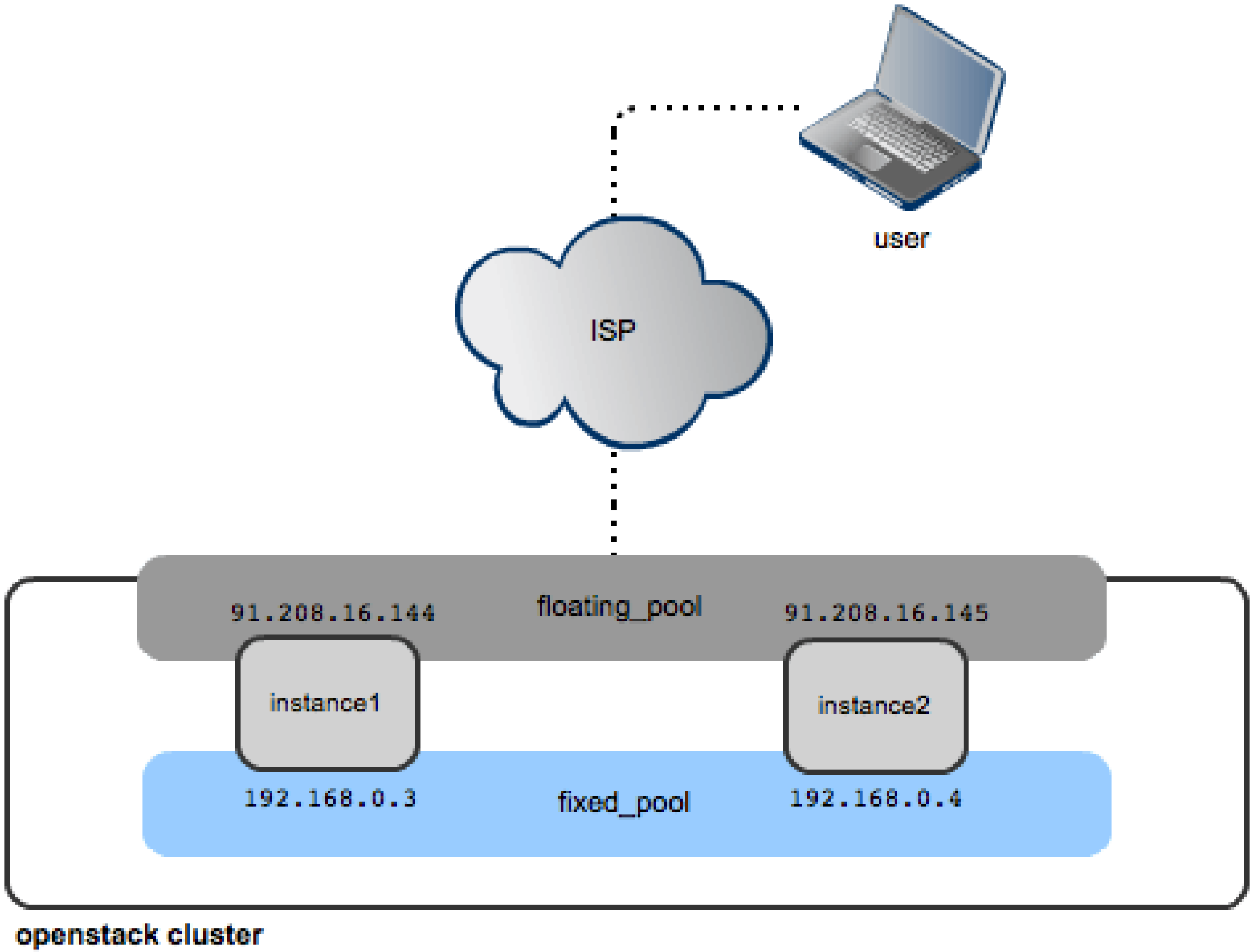
5. Connectivity from the external world

- In the previous discussion, we covered different approaches of providing Internet access to the virtual machines.
- In the **NAT-based approach**, the virtual machines can access the external world but this approach *does not allow access to the virtual machine from the Internet*.
- To expose a virtual machine to the Internet, OpenStack provides the concept of **floating IPs**.
- The **floating IP** is an external IP address that is associated with a virtual machine.
- Behind the scene, the floating IP is implemented using **DNAT.**
- This allows the virtual machine to be addressable to the external network using the associated floating IP.
- The floating IP is configured in the external interface of the router; when a packet is sent to the floating IP address, the DNAT rule configured in the router forwards the packet to the virtual machine.
- The response from the virtual machine is sent to the Internet with the source address translated to the floating IP.

6. Key features of floating IPs

- Floating IPs are not automatically allocated to instances by default (they need to be attached to instances manually).
- If an instance dies, the user can reuse the floating IP by attaching it to another instance.
- Users can grab floating IPs from different pools defined by the cloud administrator to ensure connectivity to instances from different ISPs or external networks.

- Click here for more info.


user
ISP
91.208.16.144
floating_pool
91.208.16.145
instance1
instance2
192.168.0.3
fixed_pool
192.168.0.4
openstack cluster

6. Associating a floating IP to a virtual machine

- Create an External Network:
- Creating an external network is a pre- requisite for floating IPs.
- Let's first create an external network:
  - $ neutron net–create externalJ –shared ––router:external True
    - An external network named "externalJ" is created, marked as shared and designated as an external network for routers.
- Create an External Subnet for the External Network:
  - $ neutron subnet–create externalJ ––name external–subnetJ––gateway JO.56.JJ.J JO.56.JJ.O/24
    - An external subnet named "external-subnetJ" is created for the external network.
    - The gateway for this subnet is set to 10.56.11.1 with a CIDR of 10.56.11.0/24.

6. Associating a floating IP to a virtual machine

- Set External Network as Router's Gateway:
- Next, use this network to set the gateway for your virtual router:
  - $ neturon router–gateway–set routerJ external
    - Adds a port on the external network to the router (routerJ).
    - Sets the gateway of the external network as the default gateway for the router.
- This will add a port on the external network to the router and sets the gateway of the external network as default one of the router.

- Enable Internet Access using SNAT:
- The router (routerJ) now has a default gateway set to 10.56.11.1.
- Virtual machines connected to this router can access the Internet using Source Network Address Translation (SNAT).

6. Associating a floating IP to a virtual machine

- Finally, create and associate the floating IP.
- This will make the virtual machine accessible to the Internet by associating external IP to it and implementing DNAT.
- To do this use the following command:
    - $ neutron floatingiq–create externalJ
    - $ neutron floatingiq–associate floating_iq_id instance_qort_id
    - A floating IP is created for the external network (externalJ).
    - The floating IP is associated with a specific instance by specifying the floating IP ID and the instance's port ID.
- At this point the floating IP is set and DNAT rules are added to the router namespace to forward any traffic to the floating IP address to the IP address of the specified port instance.

6. Associating a floating IP to a virtual machine

- Enable Internet Access to the Virtual Machine using DNAT:
  - DNAT rules are added to the router namespace, forwarding any traffic to the floating IP address to the IP address of the specified instance port.
  - This enables the virtual machine to be accessible from the Internet using the associated floating IP.

Computing

Demo:

Assign a Floating IP Address to an Instance in OpenStack

Click Here

05

Implementing network security in OpenStack

Module IV

Implementing network security in OpenStack

- OpenStack provides network security **to control access to the virtual networks**.
- Like other network services, the security policy applied to the virtual networks are offered as a self- service feature.
- The security services are provided **either at a network port level using security groups** or **at the network boundary using the firewall service**.

Implementing network security in OpenStack

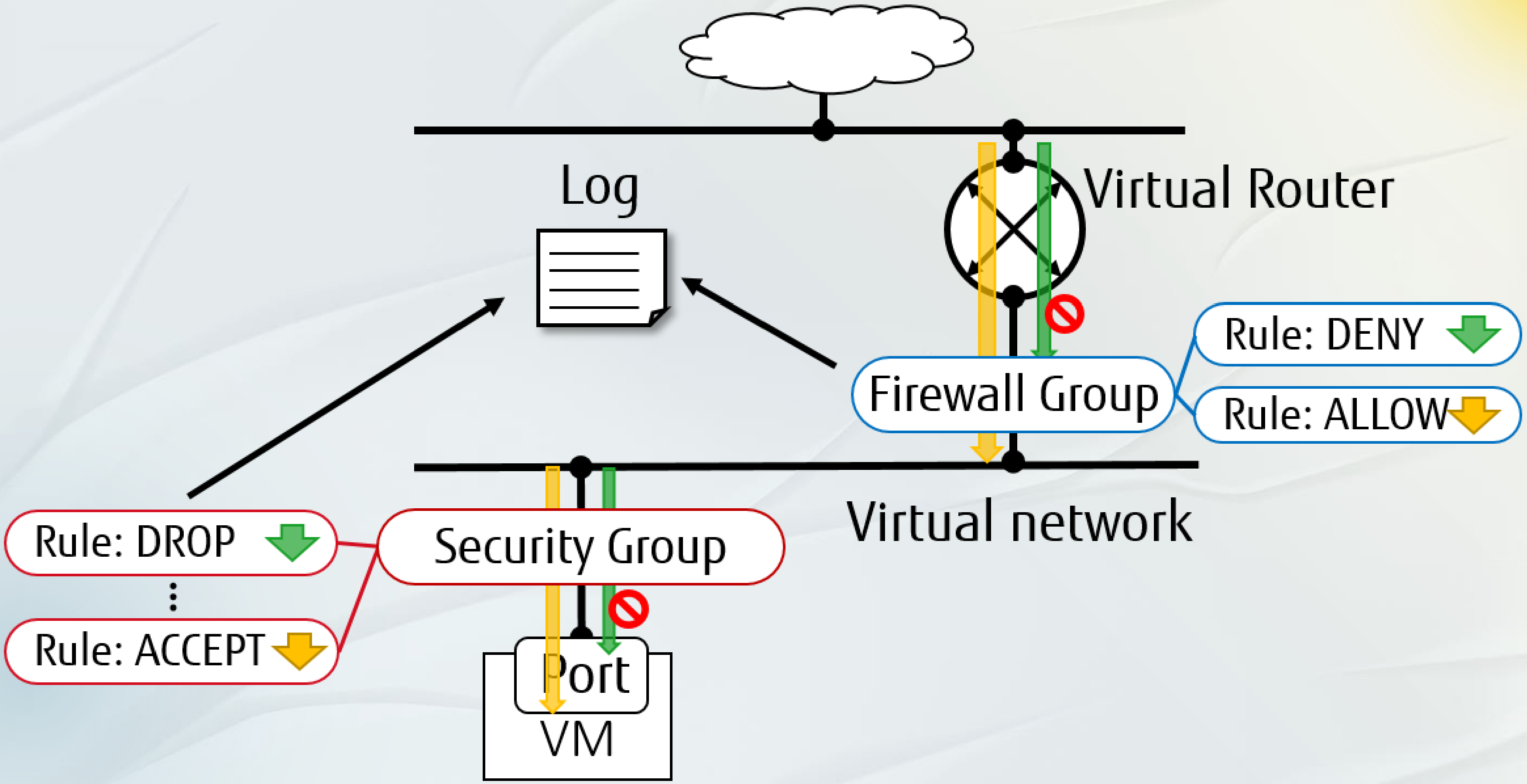
- The security rules that are applied to the incoming and outgoing traffic are based on match condition, which includes the following:
  - The source and destination addresses to which the security policy must be applied.
    - The security policy specifies the source and destination addresses to which it should be applied. This enables fine-grained control over which entities are permitted or denied access.
  - The source and destination ports for the network flow
    - Control over network flows is achieved by defining rules based on source and destination p*orts. This ensures that specific services or applications are either allowed or restricted.
  - The directionality of traffic, Egress/Ingress traffic
    - The directionality of traffic, whether it is Egress (outgoing) or Ingress (incoming), is a crucial criterion in security rule definition. This allows administrators to dictate how traffic is allowed to flow.
- The Neutron security services use Linux IPtables to implement security policies.

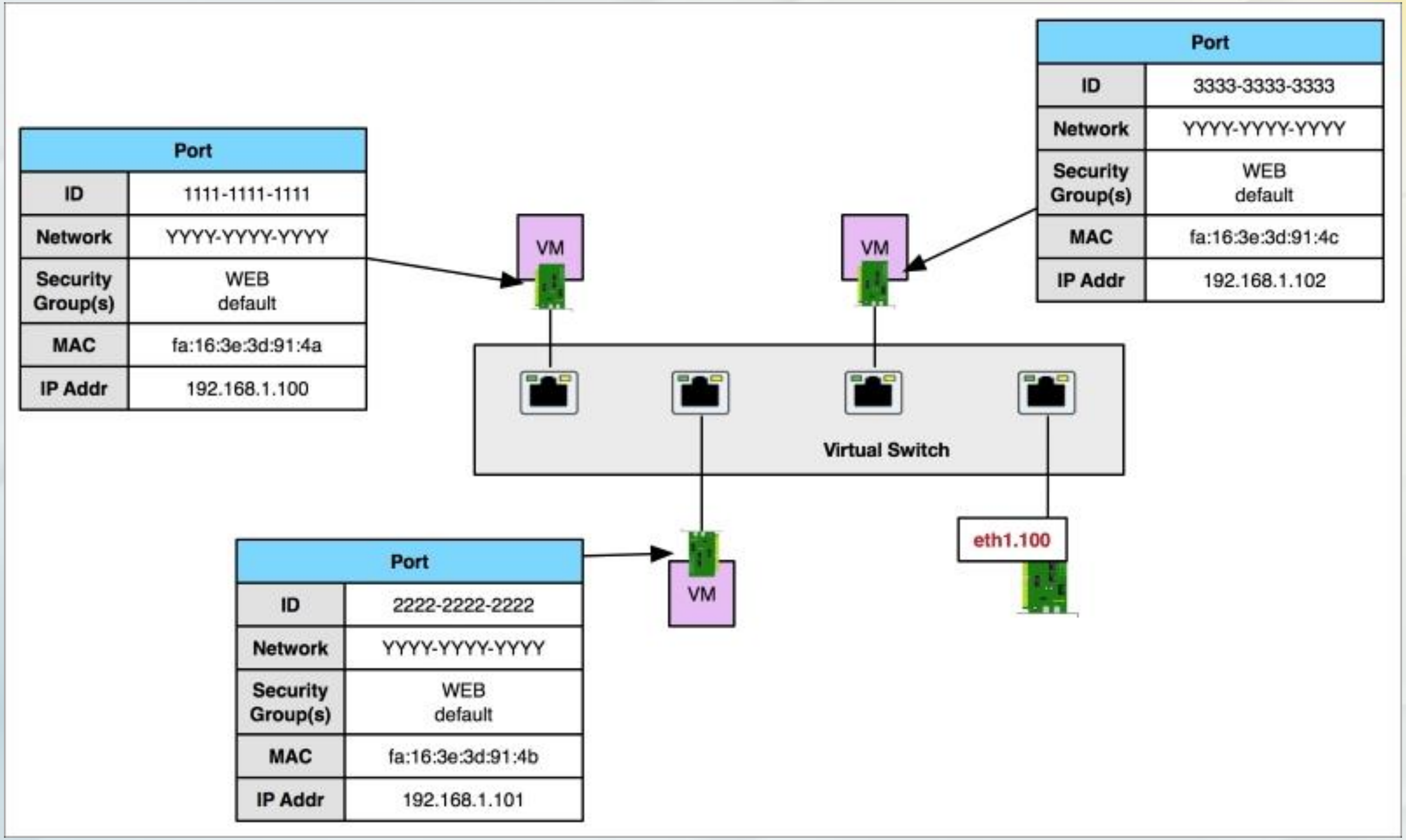
Implementing network security in OpenStack

- The Neutron security services use Linux IPtables to implement security policies.
- Key Components of IPtables Rules:
  - Match Conditions: IPtables rules are defined based on match conditions, including source and destination addresses, source and destination ports, and the directionality of traffic (Egress/Ingress).
  - Tables and Chains: IPtables organizes rules into tables (e.g., filter, nat, mangle) and chains within these tables. The rules in a chain determine the fate of a packet—whether it is accepted, dropped, or forwarded.
  - Example

```
iptables -A INPUT -p tcp --dport 22 -j ACCEPT
```

  - This rule allows incoming TCP traffic on port 22 (SSH) to be accepted.


Log
Virtual Router
Rule: DENY
Firewall Group
Rule: ALLOW
Virtual network
Rule: DROP
Security Group
Rule: ACCEPT
Port
VM



Port	
ID	1111-1111-1111
Network	YYYY-YYYY-YYYY
Security Group(s)	WEB default
MAC	fa:16:3e:3d:91:4a
IP Addr	192.168.1.100

Port	
ID	3333-3333-3333
Network	YYYY-YYYY-YYYY
Security Group(s)	WEB default
MAC	fa:16:3e:3d:91:4c
IP Addr	192.168.1.102

Port	
ID	2222-2222-2222
Network	YYYY-YYYY-YYYY
Security Group(s)	WEB default
MAC	fa:16:3e:3d:91:4b
IP Addr	192.168.1.101

Security groups

- The security groups are Neutron extensions that allow configuration of network access rules at a port level.
- With security groups, the tenant can create an access policy to control access to resources within the virtual network.
- IPtables are configured by means of security groups to perform traffic filtering.

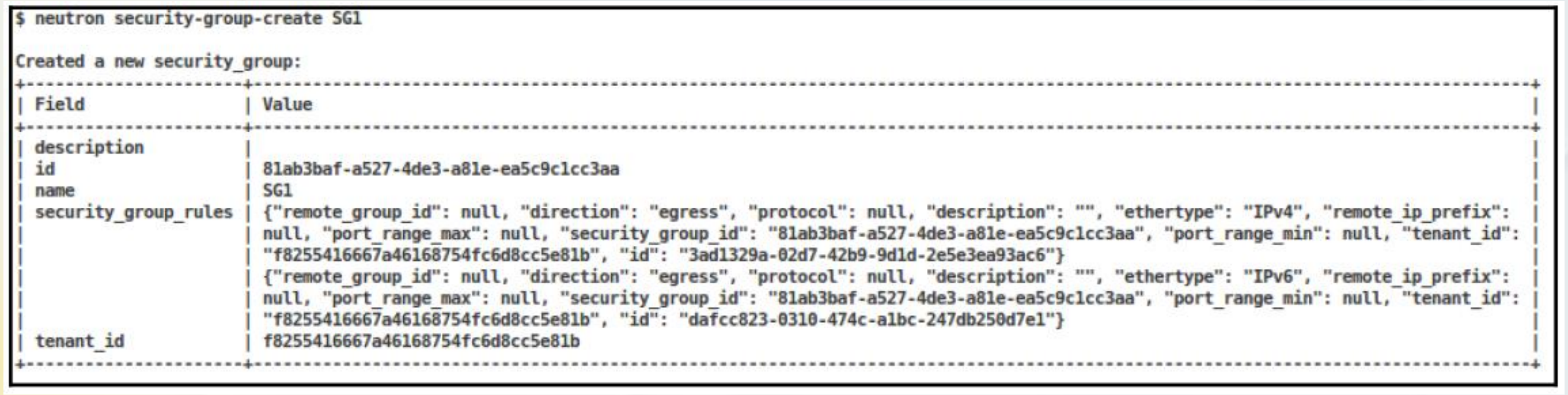
1. Creating security group policies
   - The security group rules can be created either using the Neutron CLI or through the OpenStack dashboard.
   - The **security-group-create** command is used to create a security group.
   - Subsequently, the **security-group-rule-create** command is employed to add new rules to the security group.

Security groups

CLI

neutron security-group-create <security-group-name>

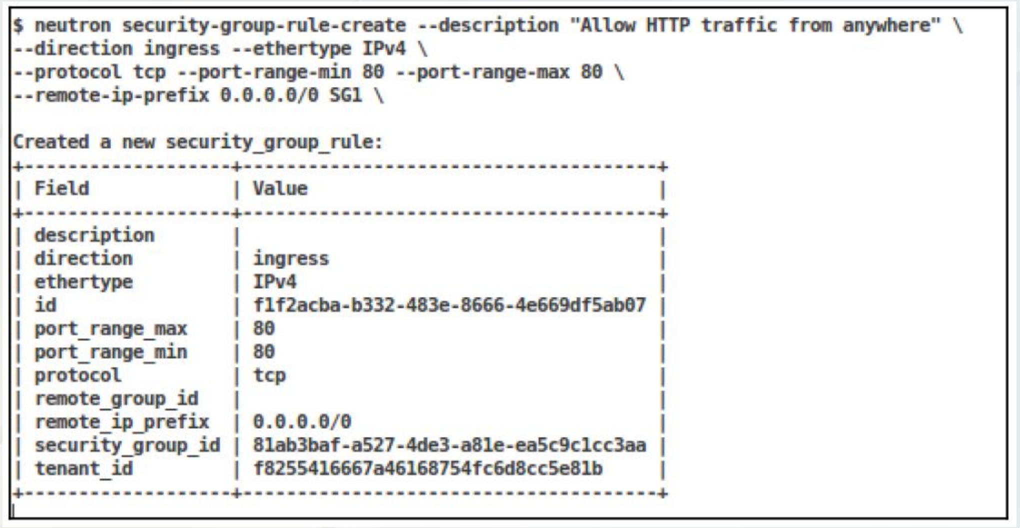
neutron security-group-rule-create --remote-address <remote-address> --protocol <protocol> --port <port> --direction <direction>



```
$ neutron security-group-create SG1

Created a new security_group:
+----------------------+---------------------------------------------------------------------------------------------------------------------------------+
| Field                | Value                                                                                                                           |
+----------------------+---------------------------------------------------------------------------------------------------------------------------------+
| description          |                                                                                                                                 |
| id                   | 81ab3baf-a527-4de3-a81e-ea5c9c1cc3aa                                                                                            |
| name                 | SG1                                                                                                                             |
| security_group_rules | {"remote_group_id": null, "direction": "egress", "protocol": null, "description": "", "ethertype": "IPv4", "remote_ip_prefix":  |
|                      | null, "port_range_max": null, "security_group_id": "81ab3baf-a527-4de3-a81e-ea5c9c1cc3aa", "port_range_min": null, "tenant_id": |
|                      | "f8255416667a46168754fc6d8cc5e81b", "id": "3ad1329a-02d7-42b9-9d1d-2e5e3ea93ac6"}                                               |
|                      | {"remote_group_id": null, "direction": "egress", "protocol": null, "description": "", "ethertype": "IPv6", "remote_ip_prefix":  |
|                      | null, "port_range_max": null, "security_group_id": "81ab3baf-a527-4de3-a81e-ea5c9c1cc3aa", "port_range_min": null, "tenant_id": |
|                      | "f8255416667a46168754fc6d8cc5e81b", "id": "dafcc823-0310-474c-a1bc-247db250d7e1"}                                               |
| tenant_id            | f8255416667a46168754fc6d8cc5e81b                                                                                                |
+----------------------+---------------------------------------------------------------------------------------------------------------------------------+
```

Security groups

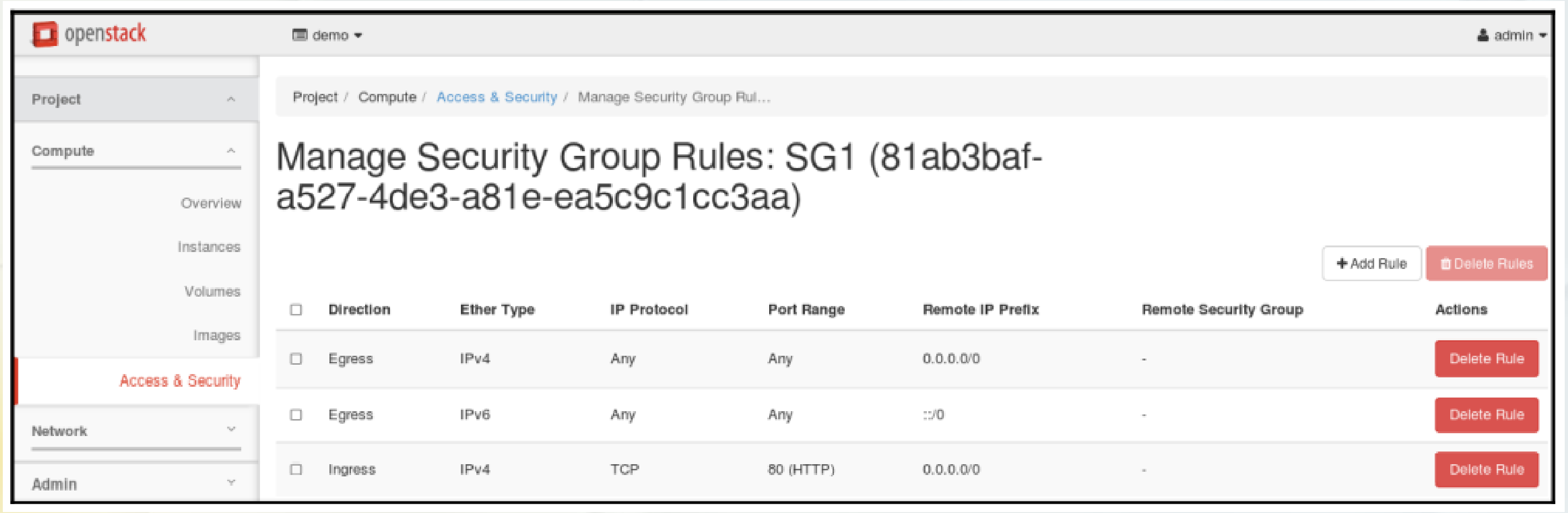
2. Then to add new rules to the security group, use the Neutron **security-group-rule- create** command.
   - To create the rule, you need to provide the remote address, the **protocol** that can be TCP, UDP, or ICMP, the protocol port, for example, 80 for HTTP traffic, and the direction of traffic, for example egress for traffic going out or ingress for traffic coming in:



```
$ neutron security-group-rule-create --description "Allow HTTP traffic from anywhere" \
--direction ingress --ethertype IPv4 \
--protocol tcp --port-range-min 80 --port-range-max 80 \
--remote-ip-prefix 0.0.0.0/0 SG1 \

Created a new security_group_rule:
+-------------------+--------------------------------------+
| Field             | Value                                |
+-------------------+--------------------------------------+
| description       |                                      |
| direction         | ingress                              |
| ethertype         | IPv4                                 |
| id                | f1f2acba-b332-483e-8666-4e669df5ab07 |
| port_range_max    | 80                                   |
| port_range_min    | 80                                   |
| protocol          | tcp                                  |
| remote_group_id   |                                      |
| remote_ip_prefix  | 0.0.0.0/0                            |
| security_group_id | 81ab3baf-a527-4de3-a81e-ea5c9c1cc3aa |
| tenant_id         | f8255416667a46168754fc6d8cc5e81b     |
+-------------------+--------------------------------------+
```

Security groups

- The same operation can also be performed using the OpenStack dashboard.
- To manage security groups, navigate to Compute | Access & Security | Security Group.
- Use the +Create Security Group button to create a new security group.
- Use the Manage Rule drop down menu to add Rules to the security group:



Project / Compute / Access & Security / Manage Security Group Rul...

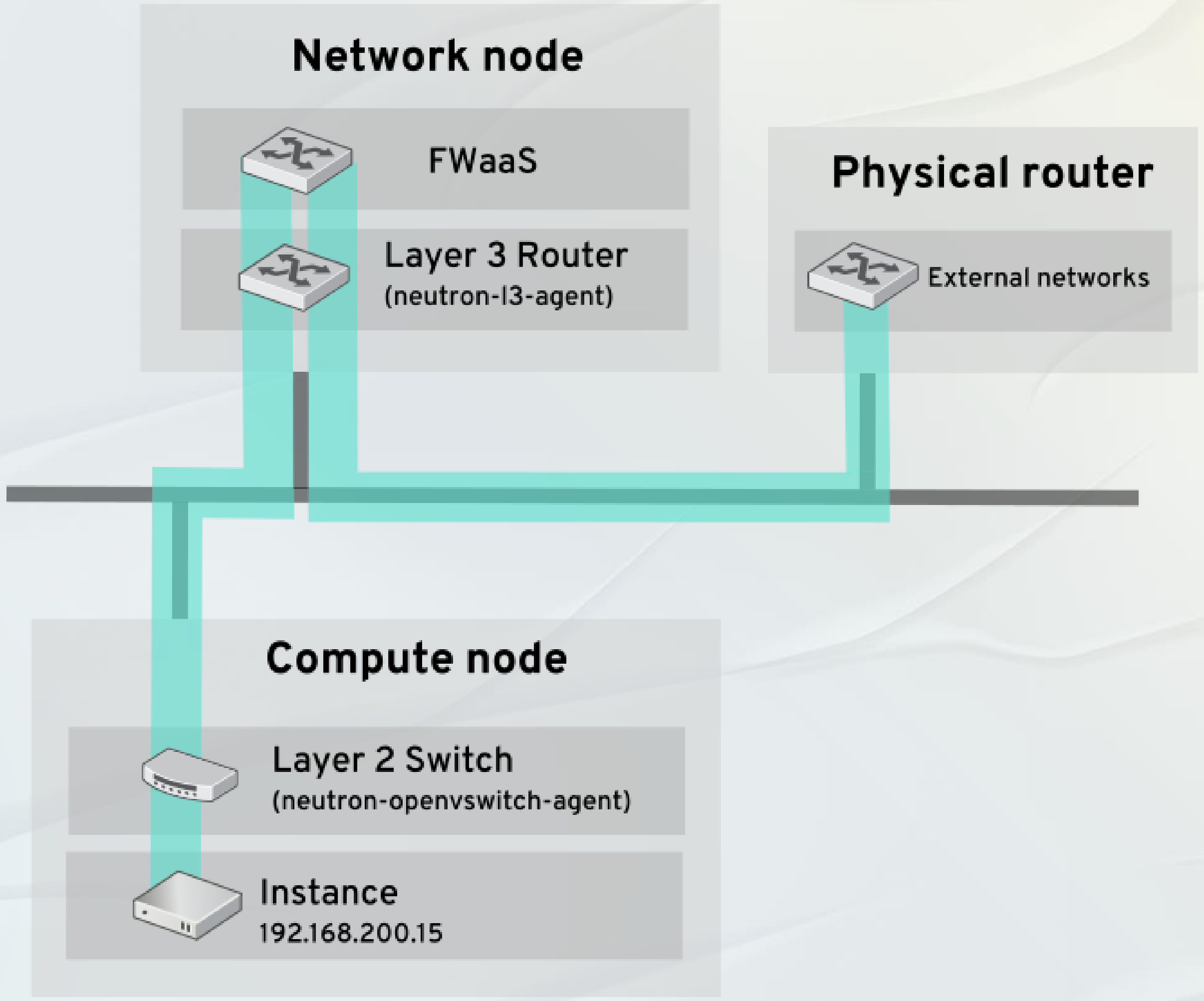
Manage Security Group Rules: SG1 (81ab3baf-a527-4de3-a81e-ea5c9c1cc3aa)

Direction	Ether Type	IP Protocol	Port Range	Remote IP Prefix	Remote Security Group	Actions
Egress	IPv4	Any	Any	0.0.0.0/0	-	Delete Rule
Egress	IPv6	Any	Any	::/0	-	Delete Rule
Ingress	IPv4	TCP	80 (HTTP)	0.0.0.0/0	-	Delete Rule

Firewall as a service

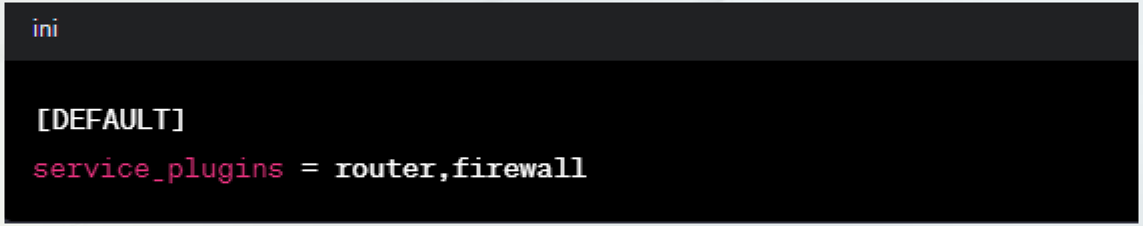
- The primary function of the firewall service is to provide control over traffic that traverses the network boundary within an OpenStack environment.
- The Firewall-as-a-Service (FWaaS) plug-in applies firewalls to OpenStack objects such as projects, routers, and router ports.
- Firewall policies are applied specifically at routers, which serve as the connection points between multiple networks.
- The default firewall driver in Neutron uses IPtables to configure access rules within the router.
- Unlike the security groups approach which is based on a default deny policy and only creates rules to allow specific traffic, the firewall service allows creation of both allow and deny rules.

1. Configuring the firewall service
2. Creating firewall policies and rules


Network node
FWaaS
Layer 3 Router
(neutron-l3-agent)
Physical router
External networks
Compute node
Layer 2 Switch
(neutron-openvswitch-agent)
Instance
192.168.200.15

1. Configuring the firewall service

1. To enable the firewall service, the Neutron configuration file (/etc/neutron/neutron.conf) needs to be updated. The following example demonstrates the necessary settings:



```ini
[DEFAULT]
service_plugins = router,firewall
```

   - firewall service plugin alongside the router service.

2. Update L3 Agent Configuration:
   - The firewall service utilizes the Neutron L3 agent to configure IPtables rules.
   - The L3 agent configuration file should be updated with the firewall driver information.
   - The following example outlines the required settings:

1. Configuring the firewall service

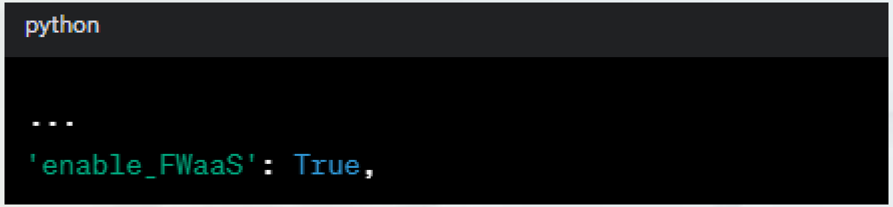
2. Update L3 Agent Configuration:
   - The following example outlines the required settings:

```
[fwaas]
driver = neutron_fwaas.services.firewall.drivers.linux.iptables_fwaas.IptablesFwaasDriver
enabled = True
```

   - This configuration specifies the firewall driver to be used (in this case, the Linux IPTables driver) and enables the FWaaS service.
3. Enable FWaaS Dashboard in Horizon:
   - To visualize and manage firewall configurations through the OpenStack dashboard (Horizon), the FWaaS dashboard must be enabled. This is done by updating the OpenStack dashboard configuration file (/usr/share/openstack-dashboard/openstack_dashboard/local/local_settings.py):

1. Configuring the firewall service

3. Enable FWaaS Dashboard in Horizon:

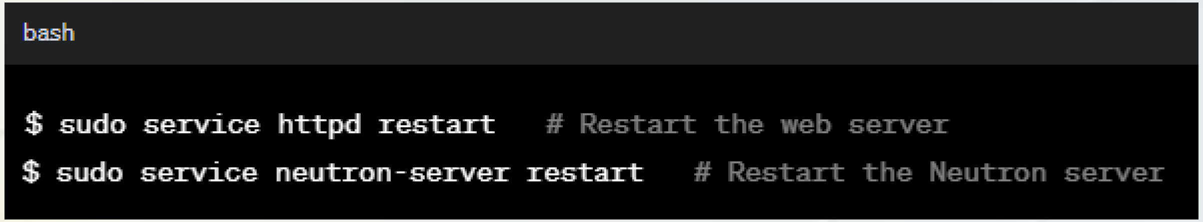


```python
...
'enable_FWaaS': True,
```

- This setting ensures that the FWaaS dashboard is activated within Horizon.

4. Restart Neutron Server and Web Server:

- After making these configuration changes, it's essential to restart both the Neutron server and the web server to apply the updates. The commands below accomplish this:



```bash
$ sudo service httpd restart   # Restart the web server
$ sudo service neutron-server restart   # Restart the Neutron server
```

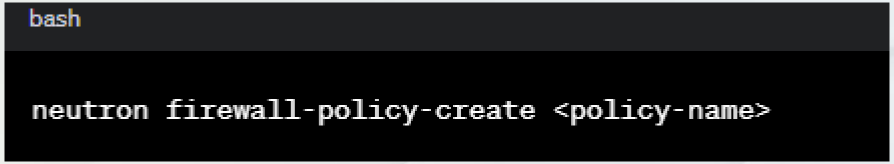
- Restarting the Neutron server loads the new service plugin configuration, and restarting the web server ensures that Horizon incorporates the changes for the firewall dashboard.

2. Creating firewall policies and rules

- The OpenStack tenant can define network access policies using the Neutron firewall service.
- The firewall policies are applied at the Layer-3 and control traffic entering or leaving a network.
- Firewall definition is composed of policies, the firewall policies themselves are composed of individual rules.
- The rules are defined based on match condition to identify flows of traffic.
- A firewall rule can explicitly allow or reject traffic flows.
- To create a firewall, use the firewall-policy-create command to create an empty firewall policy:

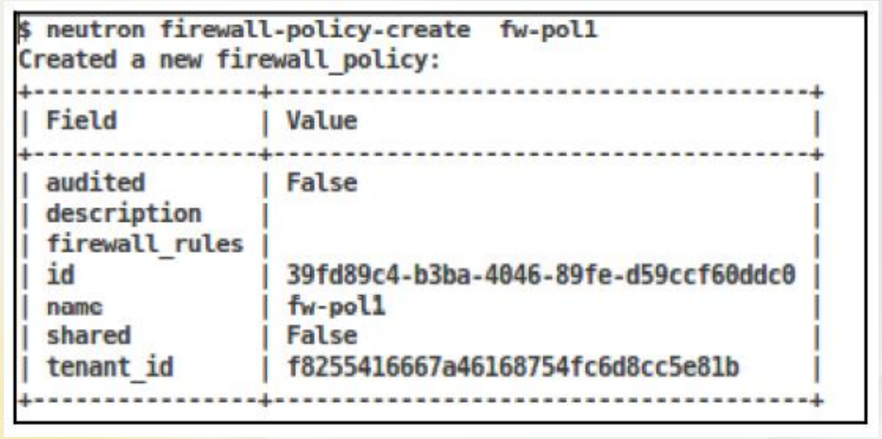
2. Creating firewall policies and rules

1. Firewall policies define the set of rules that dictate how traffic should be handled. These policies can specify both allow and deny rules.

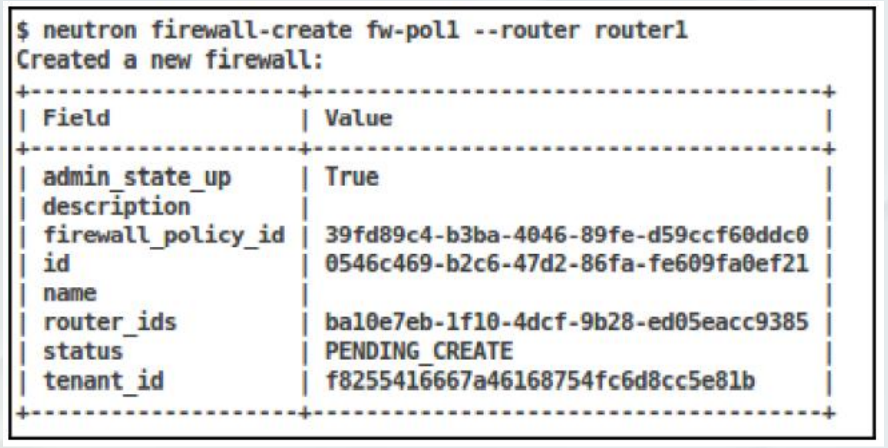


```bash
neutron firewall-policy-create <policy-name>
```

2. Then create a firewall by associating the policy created in the previous step and a router using the firewall-create command:



```
$ neutron firewall-policy-create  fw-pol1
Created a new firewall_policy:
+----------------+--------------------------------------+
| Field          | Value                                |
+----------------+--------------------------------------+
| audited        | False                                |
| description    |                                      |
| firewall_rules |                                      |
| id             | 39fd89c4-b3ba-4046-89fe-d59ccf60ddc0 |
| name           | fw-pol1                              |
| shared         | False                                |
| tenant_id      | f8255416667a46168754fc6d8cc5e81b     |
+----------------+--------------------------------------+
```

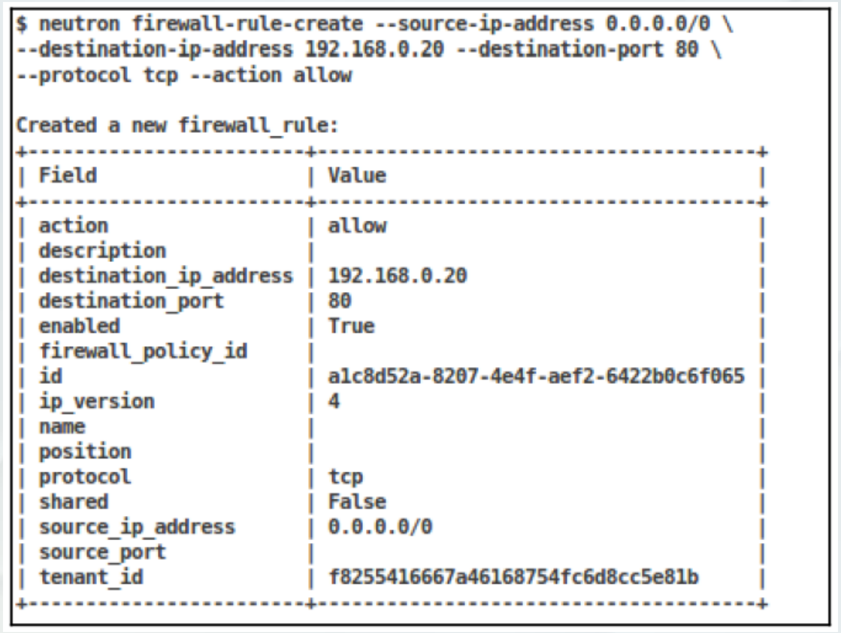


```
$ neutron firewall-create fw-pol1 --router router1
Created a new firewall:
+--------------------+--------------------------------------+
| Field              | Value                                |
+--------------------+--------------------------------------+
| admin_state_up     | True                                 |
| description        |                                      |
| firewall_policy_id | 39fd89c4-b3ba-4046-89fe-d59ccf60ddc0 |
| id                 | 0546c469-b2c6-47d2-86fa-fe609fa0ef21 |
| name               |                                      |
| router_ids         | ba10e7eb-1f10-4dcf-9b28-ed05eacc9385 |
| status             | PENDING_CREATE                       |
| tenant_id          | f8255416667a46168754fc6d8cc5e81b     |
+--------------------+--------------------------------------+
```

2. Creating firewall policies and rules

3. Firewall rules are the individual components that make up a policy.
   - They define conditions such as source and destination addresses, protocols, and actions (allow or deny).
   - Access rules can be added to the firewall policy using the firewall-rule-create command:

```
neutron firewall-rule-create --
protocol <protocol> --action
<allow/deny> --source-ip-address
<source-IP> --destination-ip-address
<destination-IP> --firewall-policy
<policy-id>
```



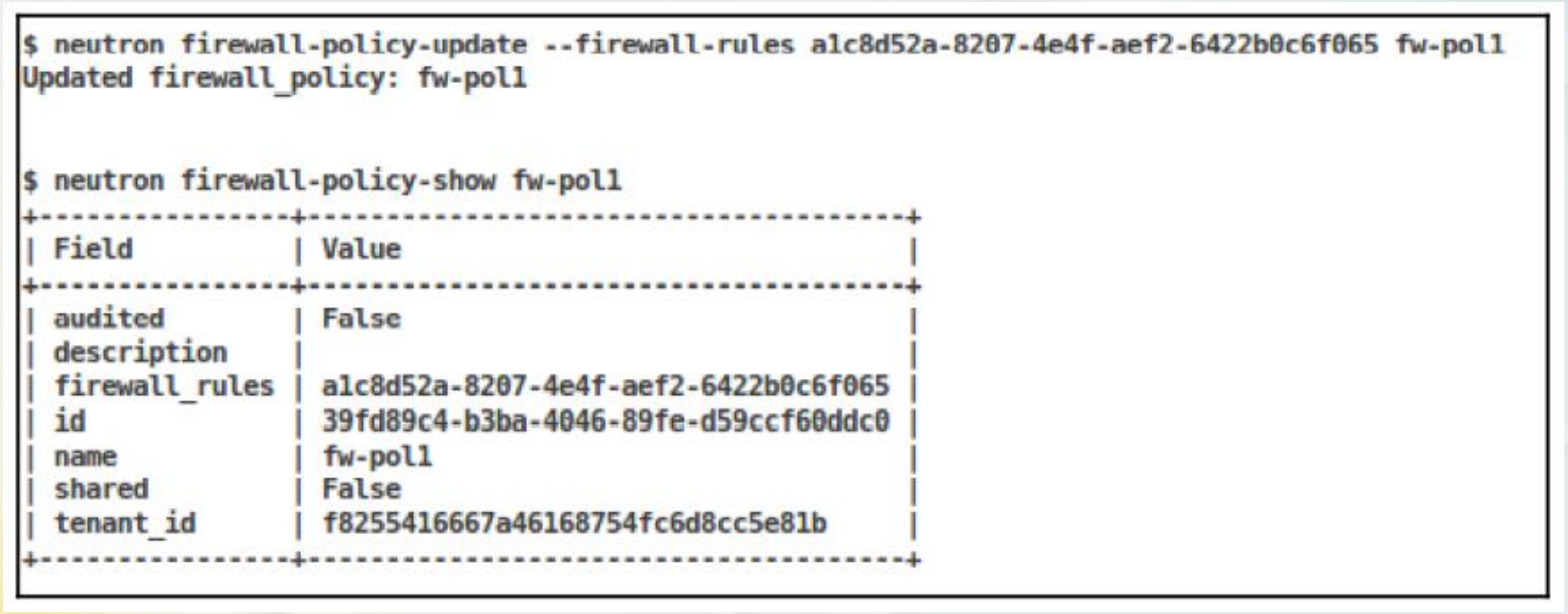
```
$ neutron firewall-rule-create --source-ip-address 0.0.0.0/0 \
--destination-ip-address 192.168.0.20 --destination-port 80 \
--protocol tcp --action allow

Created a new firewall_rule:
+------------------------+--------------------------------------+
| Field                  | Value                                |
+------------------------+--------------------------------------+
| action                 | allow                                |
| description            |                                      |
| destination_ip_address | 192.168.0.20                         |
| destination_port       | 80                                   |
| enabled                | True                                 |
| firewall_policy_id     |                                      |
| id                     | a1c8d52a-8207-4e4f-aef2-6422b0c6f065 |
| ip_version             | 4                                    |
| name                   |                                      |
| position               |                                      |
| protocol               | tcp                                  |
| shared                 | False                                |
| source_ip_address      | 0.0.0.0/0                            |
| source_port            |                                      |
| tenant_id              | f8255416667a46168754fc6d8cc5e81b     |
+------------------------+--------------------------------------+
```

2. Creating firewall policies and rules

- Then use the firewall-policy-update or firewall-policy-insert-rule command to update the firewall policy.
- The order of rules added to the firewall policy determines how packets traversing the firewall will be evaluated.
- This point must be kept in mind while providing the rule list during the firewall policy update.
- The firewall-policy- insert-rule command can be used to insert a single rule in the firewall policy.
- To identify the position of the rule in the firewall policy, use the --insert-before or --insert- after option.

2. Creating firewall policies and rules

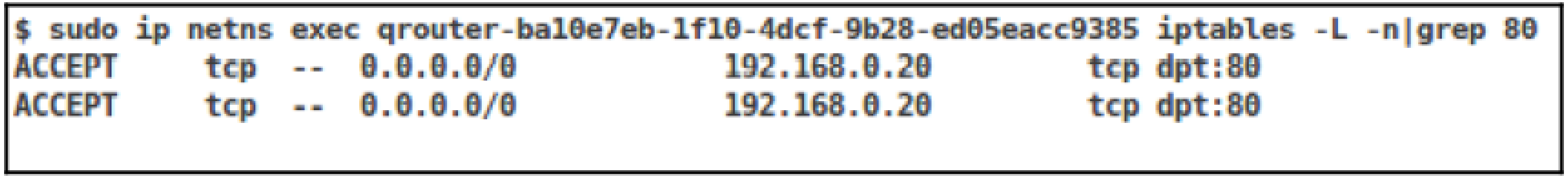
- To identify the position of the rule in the firewall policy, use the --insert-before or --insert- after option.



```
$ neutron firewall-policy-update --firewall-rules a1c8d52a-8207-4e4f-aef2-6422b0c6f065 fw-pol1
Updated firewall_policy: fw-pol1


$ neutron firewall-policy-show fw-pol1
+----------------+--------------------------------------+
| Field          | Value                                |
+----------------+--------------------------------------+
| audited        | False                                |
| description    |                                      |
| firewall_rules | a1c8d52a-8207-4e4f-aef2-6422b0c6f065 |
| id             | 39fd89c4-b3ba-4046-89fe-d59ccf60ddc0 |
| name           | fw-pol1                              |
| shared         | False                                |
| tenant_id      | f8255416667a46168754fc6d8cc5e81b     |
+----------------+--------------------------------------+
```

2. Creating firewall policies and rules

- To verify the applied firewall policy, check the IPtables configuration in the router namespace:.



```
$ sudo ip netns exec qrouter-ba10e7eb-1f10-4dcf-9b28-ed05eacc9385 iptables -L -n|grep 80
ACCEPT     tcp  --  0.0.0.0/0            192.168.0.20         tcp dpt:80
ACCEPT     tcp  --  0.0.0.0/0            192.168.0.20         tcp dpt:80
```

2. Creating firewall policies and rules

- Dashboard Integration:

- Similar to other Neutron features, configuring the firewall service can be performed not only through the CLI but also via the OpenStack dashboard.

- Operations in the Dashboard:

- Navigate to "Networking | Firewall | Policies" to create firewall policies.
- Navigate to "Networking | Firewall | Rules" to create firewall rules.
- Associate the created policy with a router.

Previous Year Question Paper

- **PART A**
- Write a short note on Neutron plugins and its categorization.
- Explain two type of Neutron subnet port connectivity.
- **PART B**
- How to implement virtual network in OpenStack and also explain two categories of implementation.
- Explain implementation of network security in OpenStack.

Previous Year Question Paper 2022

- **PART A**
- Compare core plugin with service plugin
- Explain ML2 Plugin
- **PART B**
- Explain the architecture of neutron in detail
- Summarize the two ways of implementing virtual networks with suitable diagram

Thanks!

Cloud

Do you have any questions?

navyamolkt@amaljyothi.ac.in
Youtube.com/@navyajoe