Project Undertaken During Internship (Duration 5 Week)

Internship Period: 11th June 2018 –13th July 2018

Project Done by

Atharva Muraskar

Mentored by:

Dr Munir Sayyad (Asst Vice President)

Reliance Jio Infocomm Ltd.

Cabin No. CA01, TC23, C-Wing,

Reliance Corporate Park, Thane-

Belapur Road, Ghansoli Navi Mumbai

Pin 400701.

Direct: +91-22-79689807

Mobile No.: +91-7738189979

Topics Covered:

1) Yum Server Configuration
2) DNS (Domain Name Server) configuration
3) Ntp (Network Time Protocol) configuration
4) DHCP (Dynamically Host configuration Protocol) configuration
5) OPENSTACK
6) MARIADB configuration
7)FTP (FILE TRANSFER PROTOCOL) CONFIGURATION

1. Configuration of YUM Repository

- Create a directory by using command

  #mkdir /file name

- Mount the file / directory by using command

  mount /dev/cdrom /mnt

  were mnt is the file name
- change the directory by using command

  cd /mnt
- list directory by using the command ls
- Now we have to install packages

  cd /packages

Packages are as follows:

1) vsftpd
2) python-deltarpm
3) delta-rpm
4) createrepo

- Search for packages by using command

  ls vs*

- To install package use command

  rpm –ivh vsftpd-3.0.2-9.el7.x86_64.rpm.

- make a location to save the packages say

  cd mkdir /var/ftp/pub/Server /repodata

  cd / mnt/repodata/

- copy all the packages to pub

- activate the repository

vim /etc/yum.repos.d /first.repo

In insert mode

(first repo)

name=mnt

baseurl=ftp://ip address/pub :- location of repo

192.168.137.2

enabled=1

gpgcheck=0

Coming out from insert mode (esc+wq! +enter)

- **createrepo /var/ftp/pub**
- **yum install system.config_start**

2. DOMAIN NAME SERVER (DNS)

What is DNS

The **Domain Name System** (**DNS**) is a hierarchical decentralized naming system for computers, services, or other resources connected to the Internet or a private network. It associates various information with domain names assigned to each of the participating entities. The DNS is used to resolve human readable hostnames like www.google.com into machine readable Ip address like 204.13.248.115. It also provides other information about domain names such as mail services.

Why is DNS important

- DNS is like a phone book for the internet. If you know a person's name but don't knows their telephone number, you can simply look it up in your phone book.
- To maintain address and mail exchange UpToDate to ensure that access is maintained.
- Incorrect records will prevent access to your website.
- Without DNS, you can visit websites by only its Ip address.

Configuration of DNS (Domain Name Server)

- Install DNS software

  # yum install bind

- Edit configuration file

  # vi /etc/named.conf

- Replace the IP address with your own IP address

  add a forward zone file

  Now create a file in **/var/named** directory

  Make changes in the created folder

- Start and Enable DNS services by the command

  # systemctl start dns

  # systemctl enable dns

3. NETWORK TIME PROTOCOL (NTP)

NTP is a networking protocol for clock synchronization between computer systems over packet – switched, variable – latency data networks. It is designed by David L. Mills of the University of Delaware.

NTP needs some reference clock that defines the true time to operate. All clocks are set towards that true time. It is highly scalable that network may consist of several reference clocks. It have several time sources. NTP protocol is highly accurate, using a resolution of less than a nanosecond. If connection is lost then NTP can use measurement from the past to estimate current time and error. NTP can usually maintain time to within tens of milliseconds over the public Internet and can achieve better than one millisecond accuracy in local area networks under ideal conditions.

Configuration of NTP (Network Time Protocol)

Step 1: First install and configure NTP

NTP server package is provided by default from official RHEL 7 repositories and can be installed by issuing the following command.

 # yum install ntp

Step 2:

- Open NTP daemon main configuration file for editing, comment the default list of Public Servers from **pool.ntp.org project** and replace it with the list provided by the country
- allow clients from your networks to synchronize time with this server
- To accomplish this, add the following line to NTP configuration file, where restrict statement controls, what network is allowed to query and sync time – replace network IPs accordingly.

restrict 192.168.1.0 netmask 255.255.255.0 nomodify notrap

- After editing the file with all configuration explained above save and close ntp.conf file

Step 3: After the server is installed, start NTP server, and make sure you enable it system-wide. Use the following commands to manage the service.

#systemctl start ntpd

#systemctl enable ntpd

#systemctl status ntpd

Step 4:Verify server time sync

After NTP daemon has been started, wait a few minutes for the server to synchronize time with its pool list servers, then run the following commands to verify NTP peers synchronization status and your system time.

#ntpq –p

#date –R

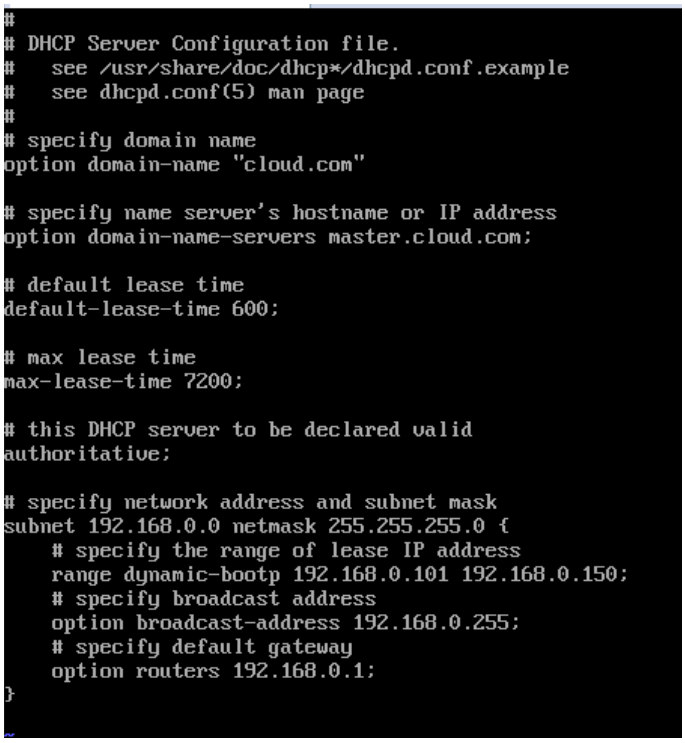
NTP IS DONE

4. Configuration of DHCP (Dynamically Host Configuration Protocol)

- First install DHCP packages. By using the code

  #yum install dhcp

- Edit dhcpd.conf file



```
#
# DHCP Server Configuration file.
#   see /usr/share/doc/dhcp*/dhcpd.conf.example
#   see dhcpd.conf(5) man page
#
# specify domain name
option domain-name "cloud.com"

# specify name server's hostname or IP address
option domain-name-servers master.cloud.com;

# default lease time
default-lease-time 600;

# max lease time
max-lease-time 7200;

# this DHCP server to be declared valid
authoritative;

# specify network address and subnet mask
subnet 192.168.0.0 netmask 255.255.255.0 {
    # specify the range of lease IP address
    range dynamic-bootp 192.168.0.101 192.168.0.150;
    # specify broadcast address
    option broadcast-address 192.168.0.255;
    # specify default gateway
    option routers 192.168.0.1;
}
```

- Start and Enable DHCP Service by using code

  # systemctl start dhcp

5. OpenStack

Introduction to OpenStack

Open stack is open source software having cloud platform. OpenStack create abstracted pools of compute, storage and networking resources that can be used to create virtual machine on the top of the hardware. Openstack is software tool that build and manage cloud computing platforms for private and public cloud.Open stack users deploy virtual machines and other instances to manage different tasks of cloud environment.

Advantages of Open stack

- **Massive Industry Support**

  Open Stack was originally formed by Rackspace and NASA in 2010 as a platform for allowing organizations to provide cloud computing services running on standard hardware. It is free and open source software managed by the OpenStack Foundation. Since its creation, an enormous number of key industry players have pledged support, including IBM, Intel, Red Hat, AMD, HP, and Canonical. Because of this strong level of investment, development moves very quickly and OpenStack has some of the most innovative companies in the world as contributors.

- **AWS Compatibility**

  OpenStack's APIs are designed to be compatible with Amazon Web Services, the most popular public cloud platform. For businesses, that means the process of porting IaaS client applications from AWS to OpenStack-based IaaS provider requires minimal effort.

- **Security**

  OpenStack has very robust role-based access controls. Access and resource utilization can be controlled at the level of users, roles, and projects. The Keystone Identity Service provides multiple forms of authentication including username /password and token-based authentication.

- **A Powerful Dashboard**

  OpenStack makes it very easy for IaaS providers to monitor and manage their cloud services. The dashboard is a web app that provides an intuitive interface for managing compute, storage, and networking resources, allowing users and administrators to have a clear overview for the management of resource usage, currently active VM instances, and users. OpenStack is a constantly evolving platform for the creation of highly available, easily managed cloud services. Businesses who intend to create their own private clouds, offer public cloud services, or use a cloud provider should strongly consider building their IaaS platform with OpenStack.

OpenStack Services

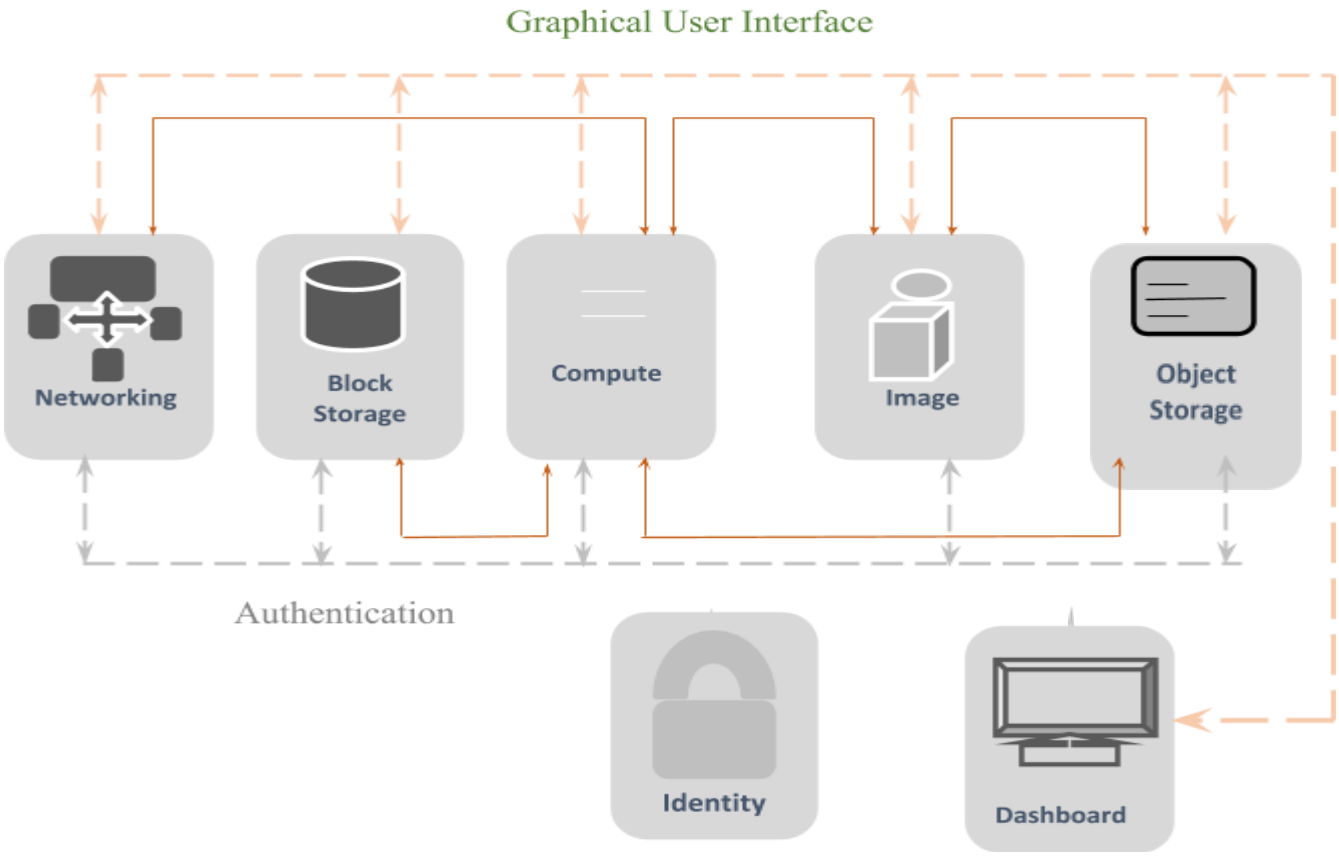
Name	Function	What it does
Horizon	Dashboard	Provides web based user interface for OpenStack services
Nova	Compute	Fetches virtual machine images from OpenStack's image service and stores on target server.
Neutron	Networking	Creates virtual network and network interfaces
Swift	Object Storage	Retrieves Objects like an image, using REST web services
Cinder	Block Storage	Provide volumes which includes disk files like logs
Keystone	Identity storage	Grants user access and process access to OpenStack tools based on keystone authentication token
Glance	Image Service	Generates VM images. Glance is a catalogue of VMs uploaded and made available to your organization.

Horizon (Dashboard)

The OpenStack dashboard is a web based user interface.It is accessed by cloud administrators and users to manage OpenStack compute, storage and networking services including Nova,Shift, and Keystone etc. via OpenStack application programming interfaces (APIs). It is based on horizon module which is series of modules called panels that define the interaction of each service. Its modules can be enabled or disabled, depending upon the service availability of the particular cloud.

OpenStack Horizon offers three versions of management dashboards:

1. User Dashboard,
2. System Dashboard and
3. Settings Dashboard.



Cloud admins are able to customize visual elements of the Horizon interface, including the navigation bar, tables, alerts and other elements. Developers can also extend any

existing dashboard to include more functionality by building an application that integrates with the dashboard.It is important to note that OpenStack Horizon is an optional application. Developers can choose to build their own user interface. Horizon is easy to style with CSS (Cascading Stylesheets).

3.3 Nova (Compute)

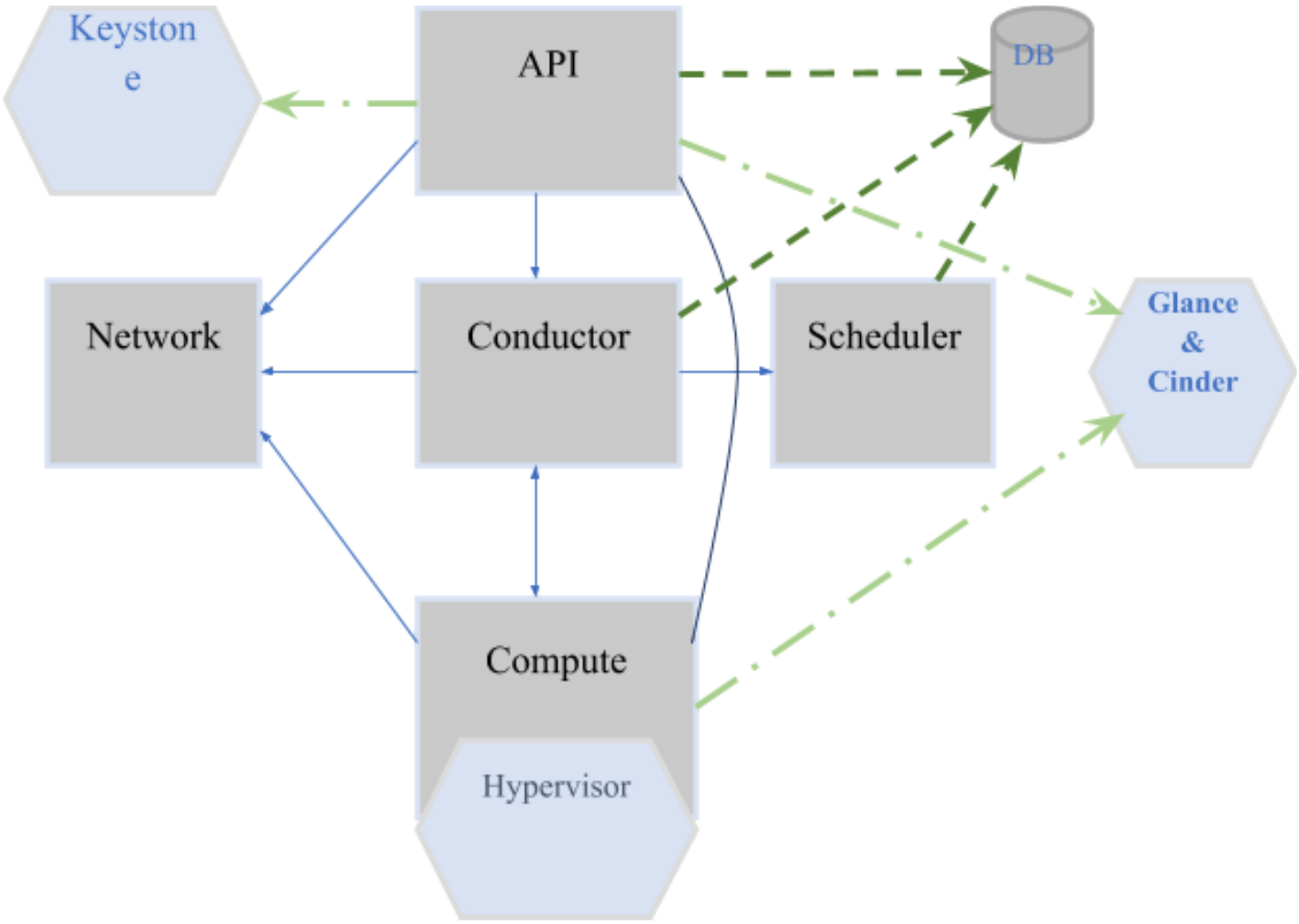
The Compute service is the heart of the OpenStack cloud by providing virtual machines on demand. Compute schedules virtual machines to run on a set of nodes by defining drivers that interact with underlying virtualization mechanisms, and exposing the functionality to the other OpenStack components. Nova is the most original core component of OpenStack; it runs a large number of requests, which are collaborated to respond to a user request into running VM.

All end user and some administrative features of nova are exposed via a REST API. The API servers process REST requests, which typically involve database reads/writes, optionally sending RPC messages to other Nova services, and generating responses to the REST calls.

Nova also uses a central database that is (logically) shared between all components. However, to aid upgrade, the DB is accessed through an object layer that ensures an upgraded control plane can still communicate with a nova-compute running the previous release. To make this possible nova-compute proxies DB requests over *RPC*to a central manager called *nova-conductor.*

Compute service components:

openstack-nova-api	Handles requests and provides access to the Compute services (such as booting an instance).
openstack-nova -compute	Creates and terminates virtual instances. Interacts with the Hypervisor to bring up new instances, and ensures that the state is maintained in the Compute database.
openstack-nova-conductor	Provides database-access support for Compute nodes (thereby reducing security risks).
openstack-nova-network	Handles Compute network traffic (both private and public access). Handles tasks as assigning an IP address to a new virtual instance.
openstack-nova-scheduler	Dispatches requests for new virtual machines to the correct node.
libvirtd	The driver for the hypervisor. Enables the creation of virtual machines.
Database	Sql database for data storage.


Keyston
e
API
DB
Network
Conductor
Scheduler
Glance
&
Cinder
Compute
Hypervisor

Neutron (Network)

Neutron provides a real Network as a Service (NaaS) between interface devices that are managed by OpenStack services such as Nova. The Neutron Allows users to manage and create networks or connect servers and nodes to various networks in the OpenStack cloud. Elements include networks, subnets, and routers; advanced services such as firewalls or virtual private networks (VPN) can also be used.

There are various characteristics that should be considered for Neutron:

- It allows users to create their own networks, control traffic and to connect server interfaces to one or more networks.
- OpenStack Networking provides cloud administrators with flexibility in deciding which individual services should run on which physical systems.
- It offers flexible networking models, so that administrators can change the networking model to adapt to their volume and tenancy.
- IPs can be dedicated or floating; floating IPs allows dynamic traffic rerouting.A floating IP address is a service provided by Neutron. It's not using any DHCP service or being set statically within the guest, would be used for accessing the instance from public networks.
- A limit of 4094 VLANs (4094 networks) can be used in OpenStack Networking. This translates to a limit of 16M tunnels in the cloud, and a limit of 4094 tunnels per compute node.

These are the main components of Neutron architecture that you ought to know of OpenStack-

Neutron-Server:

This component provides a webserver that exposes the neutron API, along with bunch of agents and plugins that communicate with each other using message queue. It passes all web services call to the neutron plugin for its action. The main function of this is to be face of the entire neutron environment to outside world.

Neutron Plugins:

The neutron-server requires indirect access to a persistent neutron database. This is accomplished through plugins, which communicate with the database using AMQP (Advanced Message Queuing Protocol).

Wide choices of plug-ins are also available. For example, the Open vSwitch and Linux bridge plug-ins use native Linux networking mechanisms, while other plug-ins interface with external devices or SDN(Software defined Network) controllers. Plug-in manages network drivers, providing routing and switching services.

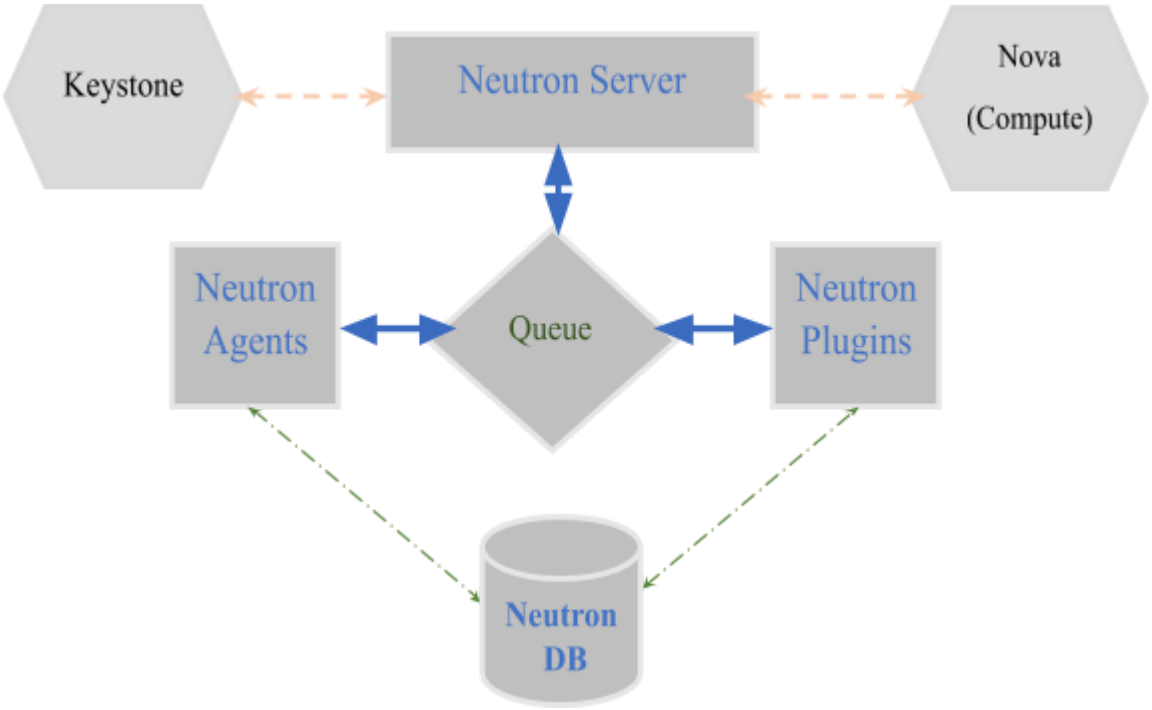


Fig 3.4 Interface Structure of Neutron (Network)

Neutron Agents:

Neutron Agents runs on each compute node to manage the service that runs on each node to perform local networking configuration for the node's virtual machines and networking services. This service requires message queue access and depends on the plugin used.

- DHCP agent (*neutron-dhcp-agent*)
  Provides DHCP services to tenant networks. This agent is the same across all plug-ins and is responsible for maintaining DHCP configuration.
- L3 agent (*neutron-l3-agent*)
  Provides L3/NAT forwarding for external network access of VMs on tenant networks.

Queue:

This routes messages between the neutron-server and various agents as well as the database to store the plugin state for a particular queue.

Keystone (Identity)

The OpenStack Identity service (keystone) is a **shared service** that provides authentication and authorization to OpenStack users; the service is used by all OpenStack components. The service supports multiple forms of authentication including user name and password credentials, token-based systems, and AWS-style logins (Amazon Web Services).

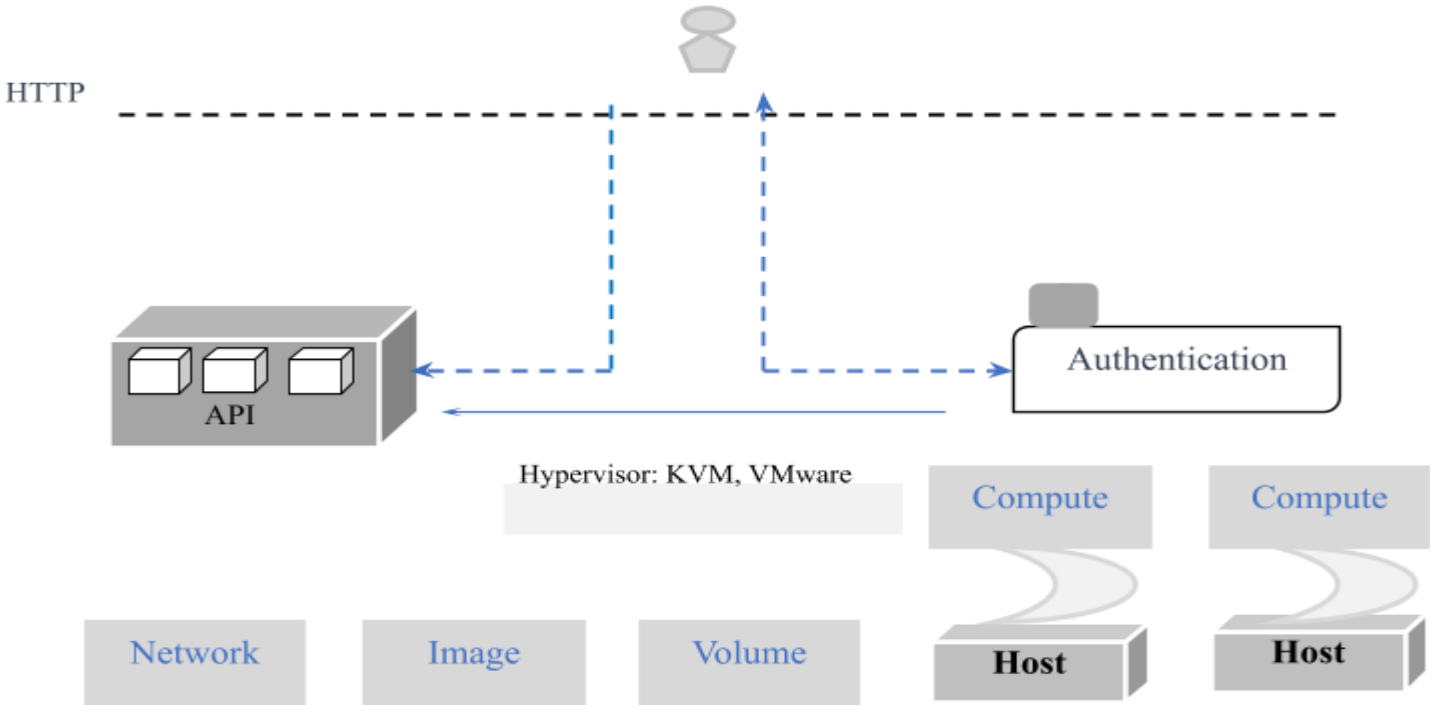


Fig 3.5 Keystone Authentication Process

The Identity service also provides a central catalog of services and endpoints running in a particular OpenStack cloud, which acts as a service directory for other OpenStack systems. Users, which have associated information such as a name and password, must be defined for each catalogued service (for example, the 'glance' user for the Image service).

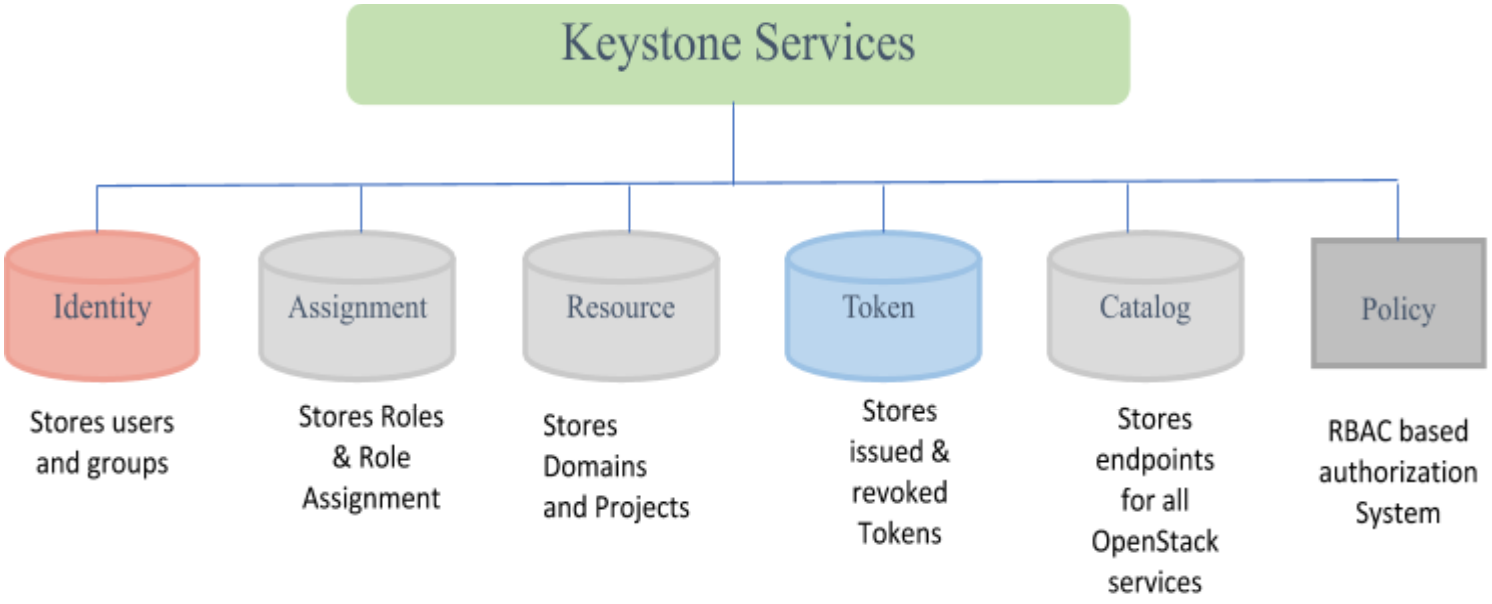


Fig 3.6 Services of Keystone

Identity service component:

openstack-keystone	Provides the administrative and public APIs.
Databases	For each of the internal services.

Project: a Project is an abstraction used by other OpenStack services to group and isolate resources (e.g., servers, images, etc.).

Domains: A Domain is formally defined as a collection of users, groups, and projects. It provides the ability to isolate the visibility of a set of Projects and Users to support multiple user organizations at the same time.

Users and User Groups: Users and User Groups are the entities given access to resources that are isolated in Domains and Projects. Groups are a collection of Users.

Identity: The Identity Service in Keystone provides the Actors.

Roles: *Roles*are used in Keystone to convey a sense of Authorization.

Assignment: It is the combination of an actor, a target, and a role which are granted and revoked, and may be inherited between groups and users and domains and projects.

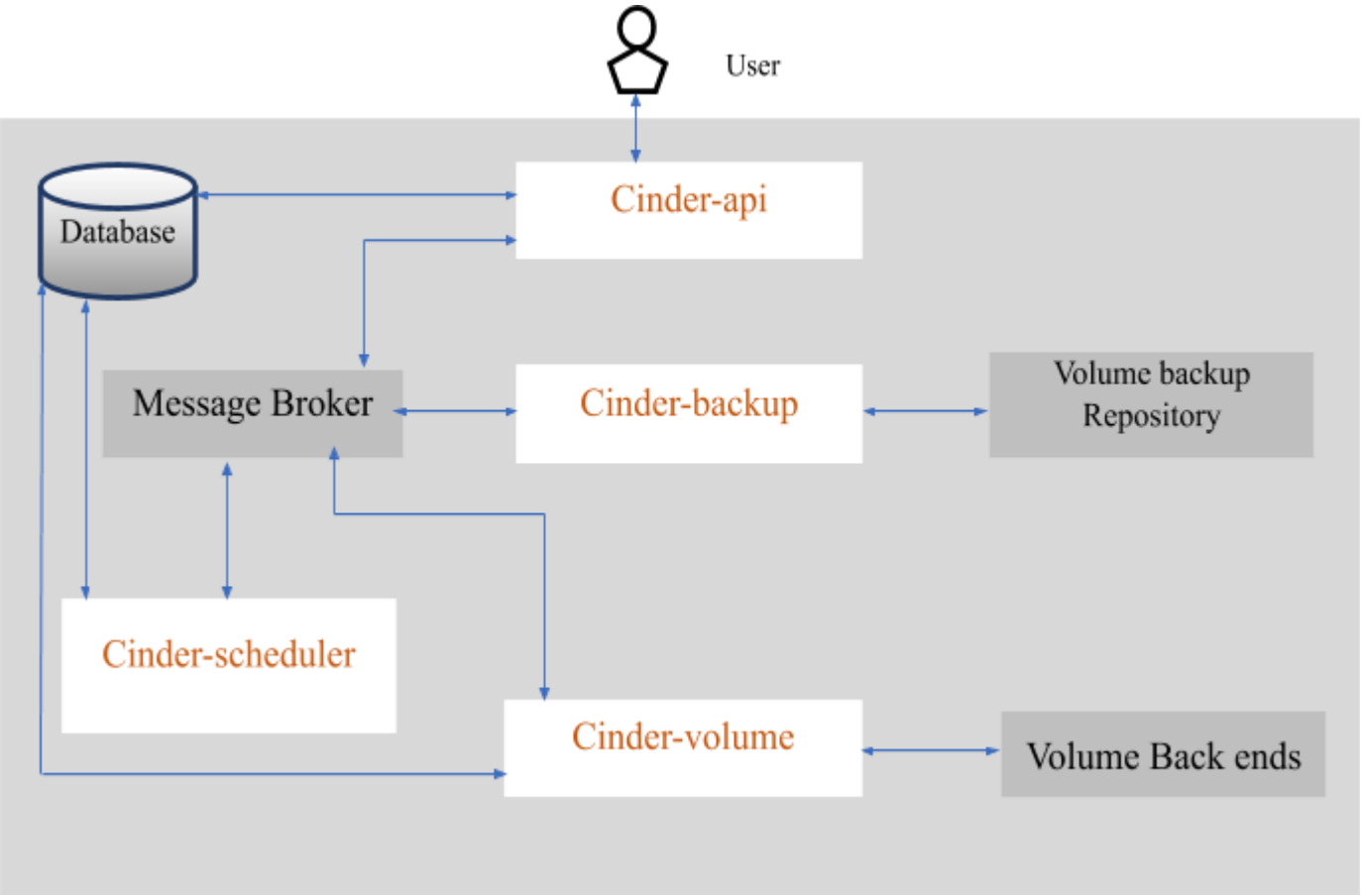
Token: The token used for authentication. It contains the authorization a user has on the cloud. A token has both an ID and a payload. The ID of a token is guaranteed to be unique per cloud, and the payload contains data about the user.

Catalog: It is a list of URLs and endpoints to route requests from different Cloud services.

Cinder (Block Storage)

Block Storage is data storage where data is stored in volumes, also referred as blocks. This service provides persistent block storage management for virtual hard drives. Block Storage allows the user to create and delete block devices, and to manage the attachment of block devices to servers.

Block storage is necessary for expandable file systems, maximum performance, and integration with enterprise storage services. The application programming interface (API) also facilitates snapshot management which can back up volumes of block storage.



Basic operations include:

- Create, list, and delete volumes.
- Create, list, and delete snapshots.
- Attach and detach volumes to running virtual machines.

The basic resources offered by the Block Storage service are volumes and snapshots which are derived from volumes and volume backups:

- Volumes: Allocated block storage resources that can be attached to instances as secondary storage or they can be used as the root store to boot instances.
- Snapshots: It is read-only copy of volume which can be created from a volume that is currently in use. Snapshots are used to create new volumes.
- **Backups**: An archived copy of a volume currently stored in OpenStack Object Storage (Swift).

The Block Storage service provides:

Cinder-api: Responds to handles requests, and places them in message queue. When an incoming request is received, the api service verifies identity requirements are met. This request is translated into message by it and sent to message broker for required actions.

Cinder-backup: Provides the ability to backup block storage volume to OpenStack object storage (Swift).

Cinder-volume: The volume service carves out storage for virtual machines on demand. It creates, modifies, and removes volumes as required.

Cinder-scheduler: The scheduler service schedules and routes requests to the appropriate volume service viz. Reads requests from the message queue and determines on which block storage action should be taken.

Database: Provides state information

Volume Back-end: The Block Storage service requires some form of back-end storage that the service is built on. The default implementation is to use LVM on a local volume group named cinder-volumes.

Swift (Object Storage)

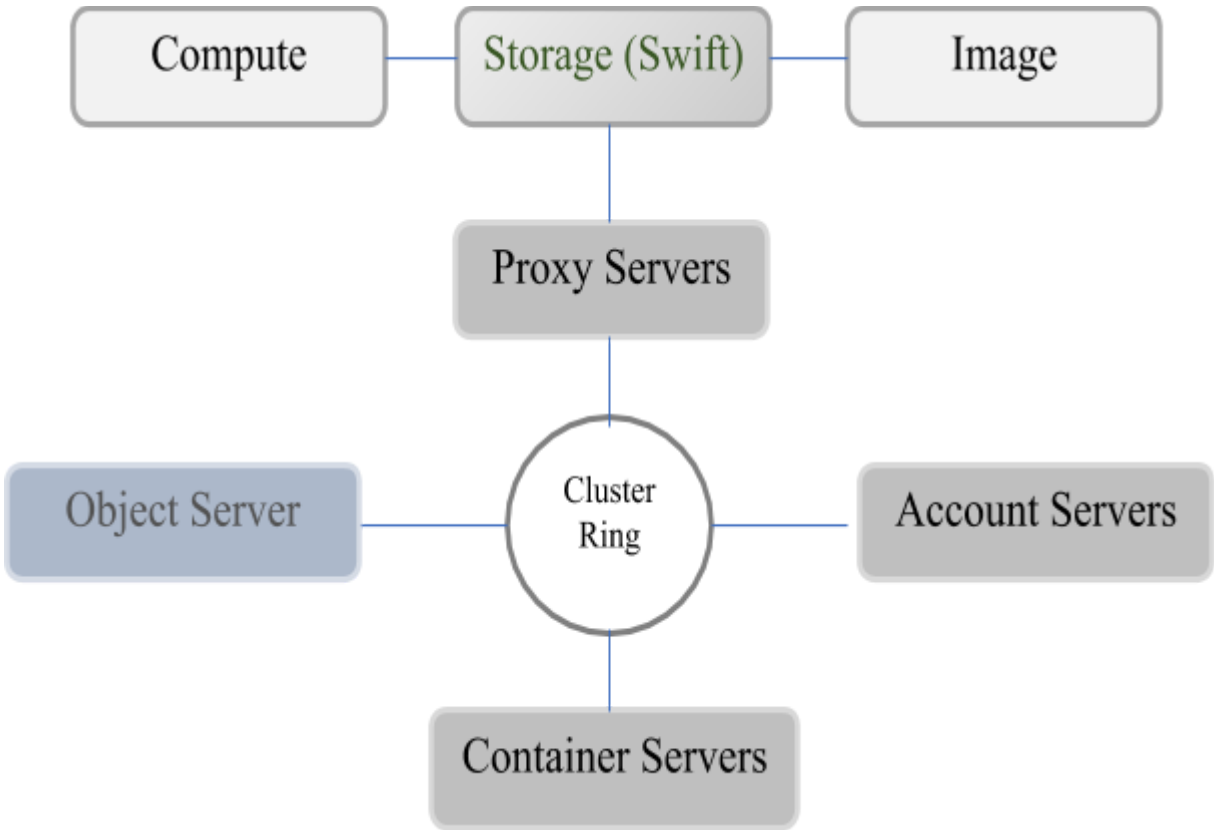
The Object Storage service provides a storage system for large amounts of data, accessible through HTTP. Static entities such as videos, images, emails, files, or VM images can all be stored. Objects (blobs of data) are stored in an organizational hierarchy that offers anonymous read-only access.
Block storage systems and filesystems are strongly consistent, which is required for databases and other real-time data, but limits their scalability and may reduce availability to data when hardware failures occur. By contrast object storage systems are eventually consistent where objects are protected by storing multiple copies of data so that if one node fails, the data can be retrieved from another node. Even if multiple nodes fail, data will remain available to the user.

Object Storage uses the concept of:

- An object is the primary storage entity. It includes the content and any optional metadata associated with the files stored in the OpenStack Object Storage system. The data is saved in uncompressed and unencrypted format and consists of the object's name, its container, and any metadata in the form of key-value pairs.
- Storage replicas, which are used to maintain the state of objects in the case of outage. A minimum of three replicas is recommended.
- Storage zones, which are used to host replicas. Zones ensure that each replica of a given object can be stored separately. A zone might represent an individual disk drive or array, a server, all the servers in a rack, or even an entire data center.
- Storage regions, which are essentially a group of zones sharing a location. Regions can be, for example, servers or server farms usually located in the same geographical area. Regions have a separate API endpoint per Object Storage service installation, which allows for isolated separation of services

.
The Swift architecture consists of the following components,
Proxy server: It accepts requests to create containers, upload files, or modify metadata and can also provide container listings or present stored files. When it receives a request, it determines the location of the account, container, or object in the ring and forwards the request to the relevant server.

Object Server: An object server is a simple server that can upload, modify, and retrieve objects (usually files) stored on the devices it manages.



Container Server: It handles the assignment of objects to a specific container and provides listings of the containers on request. The listings are replicated across the cluster to provide redundancy.

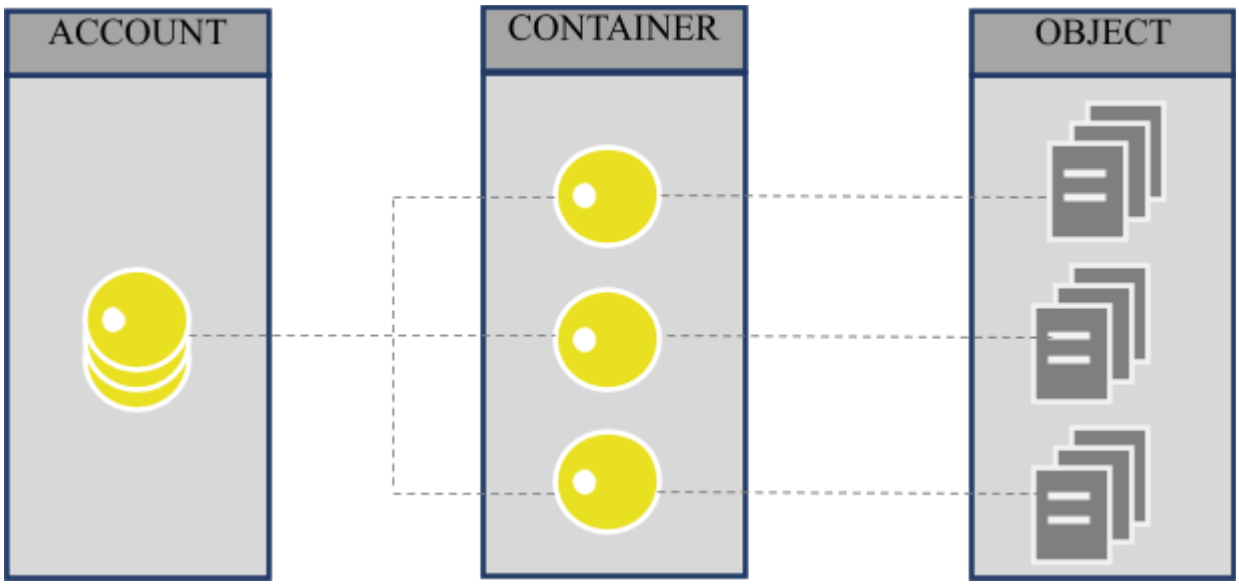
Account Server: An account server manages accounts by using the object storage services. It operates similarly to a container server in that it provides listings of containers.

Cluster Ring: Contain details of all the storage devices, and are used to map the names of stored entities to their physical location. One file is created for each object, account, and container server.

Some of the characteristics of Swift,

- The object storage location is where the data object and its metadata will be stored and all objects have a URL through which they can be indexed and searched.

- Swift runs on standard Linux distributions and on standard x86 server hardware.
- Swift is scaled by adding additional nodes, which allows for a cost-effective linear storage expansion.



- When adding or replacing hardware, data does not have to be migrated to a new storage system, i.e. there are no fork-lift upgrades.
- Swift can be accessed with HTTP requests directly to the API or by using one of the many Swift client libraries such as Java, Python, Ruby, or JavaScript.

Glance (Image)

The image service (Glance) is an 'image store' system. So what is Image store? Image is a single file which contains a virtual disk that has a bootable operating system installed on it. And an image store is where the virtual machine images managed by Glance reside on a persistent medium.

However, the difference between glance and swift is that swift stores data such as virtual disks, images, backup archiving and so forth whereas glance stores metadata. Metadata is information about kernel, disk images, disk format, etc.

Glance image services include discovering, registering, and retrieving virtual machine (VM) images. Glance has a restful API that allows querying of VM image metadata as well as retrieval of the actual image.

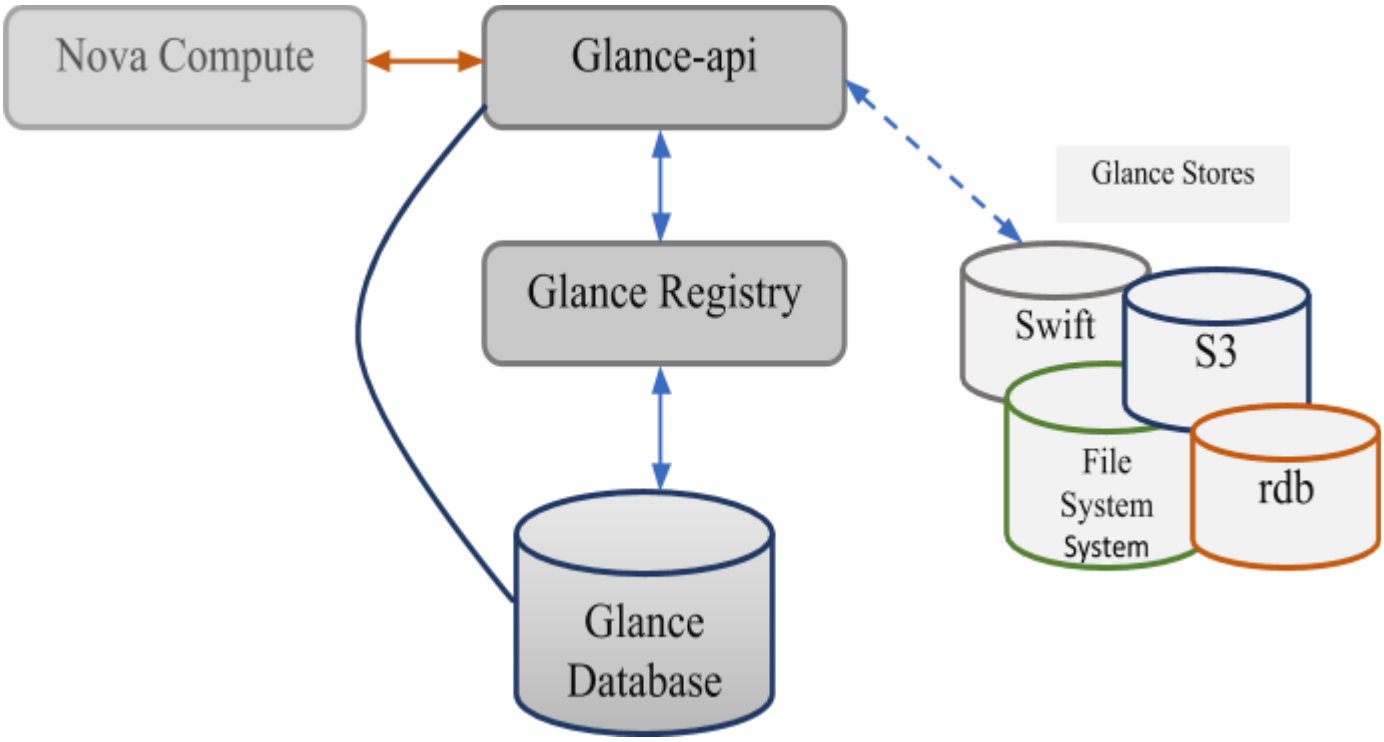


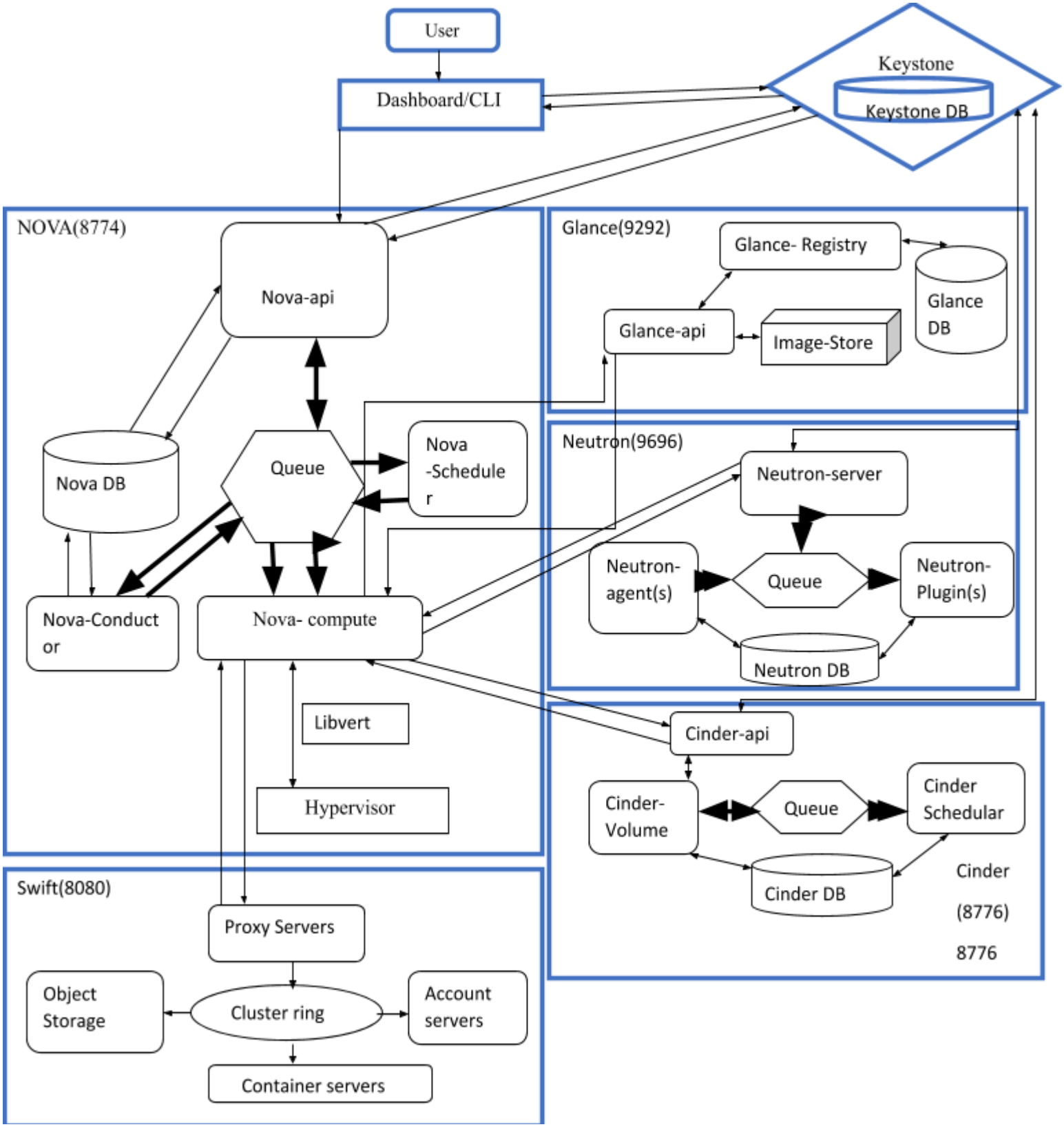
Image service components:

- **Glance-api:** Handles requests and image delivery and uses the registry to retrieve image information (the registry service is should never be, accessed directly).
- Glance-registry: stores, processes and retrieves metadata associated with each images (size, type, etc.).
- Database: Stores image metadata
- **A storage repository:** for the actual image files. In many deployments, this is OpenStack Swift.Registered images can be stored in the Object Storage service, as well as in other locations (for example, in simple file systems or external web servers).

The following image formats are supported:

- raw (unstructured format)
- aki/ami/ari (Amazon kernel, RAM disk, or machine image)
- iso (archive format for optical discs; for example, CD)
- qcow2 (Qemu/KVM, supports Copy on Write)
- vhd (Hyper-V, common for virtual machine monitors from VMware, Xen, Microsoft, VirtualBox, and others)
- vmdk (VMware)

Request flow for provisioning Instance in Open stack



1. User accesses the Openstack environment via the web interface (HTTP/REST). User login into the dashboard/CLI using their own credentials i.e. username and password.
2. Authentication is the first action performed by the keystone where user provides its credentials. After verifying the credentials in keystone db (database where the all user records are stored) and the user is valid it sends the authentication success message, authentication token and service catalog list. If the user is invalid then the authentication request failed. In case of the new user it sets the credentials and record the entry of the new user in the keystone db.
3. After successful authentication user can talk to an API node. The users request and authentication token generated by the keystone are provided to the nova-api of the NOVA component. Users request contain Flavor, Image, Network and VM name. In flavor there is the requirement of disk, core and RAM.
4. After receiving the request to the nova-api it verifies with keystone that the user is valid or not.
5. Keystone validates if it is valid user and sends the success message to the nova-api and if the user is invalid it will discard the request.
6. Nova-api stores the request in Nova DB. Nova DB is the SQL database for storing data.
7. After storing the request in Nova DB, Nova API takes the database entry ID and request ID from the Nova DB.

8. Nova API puts the DB ID and request ID in the Queue. Queue provides the central HUB for the communication between the different components of Nova.
9. Nova scheduler always reads queue and takes the DB ID and request ID which is responsible for the VM instance creation and also determines where it should be done.
10. Using the DB ID Nova scheduler interacts with the Nova DB.
11. After interacting with the Nova DB Nova scheduler knows the flavor details which includes disk, core and RAM.
12. Receiving the flavor detail Nova scheduler identify the correct compute node to create virtual machine.
13. Nova compute component creates and terminates VM instances. Nova compute accepts DB ID and Request ID from the queue.
14. Nova compute places DB ID in the conductor. Nova compute can not directly communicate with the Nova conductor, it communicates through the queue. And also Nova compute doesn't have access to Nova DB because both are running on different servers.
15. Nova conductor takes the details from the queue i.e. the DB ID.
16. Nova conductor sends that DB ID to the Nova DB.
17. Using the DB ID Nova DB extracts the details of the request and send it to the Nova conductor.
18. Nova conductor places all the details in the queue.
19. Now the Nova compute communicates with the Glance module in openstack. Glance is an image repository, in other words, it stores the image templates and provide it to NOVA compute when requested for it.

Nova compute request via rest api to fetch image also it sends the image details and authentication token to glance-api.

20. Glance api first validate the authentication token send by the Nova compute with keystone. Image store is the middleware between Glance and the other storing devices. Glance registry processes and manipulates the metadata of the objects containing the information of the physical location of the images. This metadata is stored in the Glance Database. The Glance Database then provides this information to locate the exact image in physical memory.
21. The glance api provides the exact location of the image to the nova compute. Thus the nova compute gets the image details also.
22. For the requirement of the network nova compute communicates with the Neutron(Quantum). Nova compute sends the required details to the quantum-server along with the authentication token.
23. Before communication quantum-server validate the authentication token with the keystone. OpenStack Networking allows you to create and manage network objects, such as networks, subnets and ports, which other OpenStack services can use. The openstack networking api server supports the layer 2 networking and IP address management. Quantum plugins can provide the IP addressing. Queue accepts and routes RPC requests between agents to complete API operations.
24. Neutron provides the required details to nova compute.
25. Now nova compute communicate with Cinder which is the block storage provider. Nova compute sends the storage request details along with the authentication token to the cinder-api.

26. Cinder API validates the authentication token with the keystone. Cinder API authorizes and routes user requests through the Block Storage service .Cinder client requests via the REST API to cinder api which then processes and sends the requests via Advanced Message Queuing Protocol (AMQP) to the respective destinations. Cinder Scheduler schedules and routes requests to appropriate volume services. Algorithms involved in scheduling are Round Robin or Filter Scheduler depending on the infrastructure used.
27. After processing the request by the cinder it gives to the Nova compute.
28. Nova compute communicate with the swift module of openstack for the block storage.
29. Up to this Nova compute have the flavor, image and network details which are further responsible for the VM creation. Now the virtual machine should have to place on the hypervisor layer. Hypervisor is a firmware/ software that provide the link between hardware resources and the virtual machine. The virtual machine cannot communicate with the hypervisor directly, it requires the libvirt component. Libvirt is the open source api and management tool for managing the platform virtualization.

OpenStack Releases

OpenStack release	Component	Feature	Release Date and status
Austin	Nova, Swift	Install from package, Images without Ram disk, Object size	21 October 2010 End of Life)
Bexar	Nova, Glance, Swift	Allow users to pre-install and create their own application environments and instantly spin up the same copy as they scale out.	3 February 2011 (End of Life)
Cactus	Nova, Glance, Swift	Compute, object storage, and the new image service with a focus on stability, reliability for larger scale cloud deployments	15 April 2011 (End of Life)
Diablo	Nova, Glance, Swift	Unique in many regards in signalling that OpenStack is delivering on its potential to become the de- facto standard in cloud OS	22September 2011 (End of Life)
Essex	Nova, Glance, Swift, Horizon, Keystone	Focuses on quality, usability and extensibility across enterprise, service provider, self managed compute.	5 April 2012 (End of Life)
Folsom	Nova, Glance, Swift, Horizon, Keystone, Quantum, Cinder	Built-in-multi-tenancy, Customize at launch, Import templates	27 September 2012 (End of Life)
Grizzly	Nova, Glance, Swift, Horizon, Keystone, Quantum, Cinder	Building public, private and hybrid clouds, Native support for VMware ESX and Microsoft Hyper-V hypervisors, new ‘cells’ capabilities, provides more	4 April 2013 (End of Life)

		extensive Software Defined Networking(SDN) functionality	
Havana	Nova, Glance, Swift, Horizon, Keystone, Neutron, Cinder, Heat, Ceilometer	Support software development, managing data and infrastructure at scale, UX and interface improvements	17 October 2013 (End of Life)
Icehouse	Nova, Glance, Swift, Horizon, Keystone, Neutron, Cinder, Heat, Ceilometer, Trove	Extend Volume, Launch database, Inline Editing.	17 April 2014 (End of Life)
Juno	Nova, Glance, Swift, Horizon, Keystone, Neutron, Cinder, Heat, Ceilometer, Trove, Sahara.	Widely supported cloud platform, Storage policies, new data processing service, improved agility and efficiency in Telco and service provider data centres.	16 October 2014 (End of Life)
Kilo	Nova, Glance, Swift, Horizon, Keystone, Neutron, Cinder, Heat, Ceilometer, Trove, Sahara, Ironic	Big data analysis and application infrastructures at scale, URL allows private object storage container to be publically available for a specified period of time.	30 April 2015 (End of Life)
Liberty	Nova, Glance, Swift, Horizon, Keystone, Neutron, Cinder, Heat, Ceilometer, Trove, Sahara, Ironic, Zaqar, Manila, Designate,	Enhanced manageability, simplified scalability, Extensible to support new technologies, container management	16 October 2015 (End of Life)

	Barbican, Searchlight.		
Mitaka	Nova, Glance, Swift, Horizon, Keystone, Neutron, Cinder, Heat, Ceilometer, Trove, Sahara, Ironic, Zaqar, Manila, Designate, Barbican, Searchlight, Magnum	Enhanced end user experience, manageability, scalability	7 April 2016 (End of Life)
Newton	Nova, Glance, Swift, Horizon, Keystone, Neutron, Cinder, Heat, Ceilometer, Trove, Sahara, Ironic, Zaqar, Manila, Designate, Barbican, Searchlight, Magnum, aodh, cloudkitty, congress, freezer, mistral, monasca-api, monasca-log-api, murano, panko, senlin, solum, tacker, vitrage, Watcher	Improved scalability, Enhanced Resiliency, Expended versatility	6 October 2016 (End of Life)
Ocata	Nova, Glance, Swift, Horizon,	Ease of deploying, managing and upgrading container orchestration, improved upgrades	22 February 2017 (Maintained)

	Keystone, Neutron, Cinder, Heat, Ceilometer, Trove, Sahara, Ironic, Zaqar, Manila, Designate, Barbican, Searchlight, Magnum, aodh, cloudkitty, congress, freezer, mistral, monasca-api, monasca-log-api, murano, panko, senlin, solum, tacker, vitrage, Watcher	and key enhancement offer better performance at scale.	
Pike	Nova, Glance, Swift, Horizon, Keystone, Neutron, Cinder, Heat, Ceilometer, Trove, Sahara, Ironic, Zaqar, Manila, Designate, Barbican, Searchlight, Magnum, aodh, cloudkitty, congress, freezer, mistral, monasca-api, monasca-log-api, murano, panko, senlin, solum,	Python 3.5 support, cinder:revert to snapshot, extend volume while attached, Ironic: rolling upgrade support, Swift: Support for globally distributed erasure codes, standalone cinder, Enhanced Ironic and Neutron integration.	30 August 2017 (Maintained)

	tacker, vitrage, Watcher		
Queens	Aodh, Barbican, blazer, Ceilometer, Ceilometer-power vm, Cinder, Cloudkitty, Congress, Designate, Freezer, Glance, Heat, Horizon, Ironic, Karbor, Keystone, Magnum, Manila, Mistral, Monasca-api, Monasca-log-api, Murano, Neutron, Nova, Octavia, Panko, Sahara, Searchlight, Senlin, Solum, Storlets, Swift, Tacker, Tricircle, Trove, Vitrage, Watcher, Zaqar, Zun.	/AI/Machine learning, high availability, Enterprise workloads, Edge computing, Operator Enhancement.	28 February 2018 (Maintained)
Rocky	Barbican, Blazar, Cinder, Congress, Cyborg, Designate, Freezer, Glance, Heat, Horizon, Keystone, Manila, Masakari, Mistral, Monasca-api,	Expected to add a host of new and enhanced capabilities to the open source cloud platform, modify configuration option without a service restart.	30 August 2018 (Development)

	Neutron, Nova, Octavia, Qinling, Sahara, Searchlight, Senlin, Storlets, Swift, Trove, Vitrage, Watcher, Zaqar.		
Solar	Future		

Openstack configuration

So, I am using CentOS 7 in VMware machine. You can also use RHEL 7 (Red Hat Enterprise Linux 7) for installing/configuring OpenStack.x86_64 is currently the only supported architecture.

Prerequisites: -

- Minimum recommended versions – RHEL, CentOS 7, Scientific Linux, Ubuntu 16.04 LTS, etc.
- CPU requirements – Hosts should have Multicore processors with hardware-assisted virtualization extensions.
- RAM requirements – 4gb for better performance.
- Disk size requirements – 20gb minimum.

Installation: -

If you are using the external network access to server, then we have to configure your network settings.

- we have to disable our firewall.

```
# systemctl disable firewalld
```

```
[root@osp9 ~]# systemctl disable firewalld
Removed symlink /etc/systemd/system/multi-user.target.wants/firewalld.service.
Removed symlink /etc/systemd/system/dbus-org.fedoraproject.FirewallD1.service.
```

- Then stop the firewall

```
# systemctl stop firewalld
```

```
[root@osp9 ~]# systemctl stop firewalld
[root@osp9 ~]#
```

- Disable our Network Manager

```
# systemctl disable NetworkManager
```

```
[root@osp9 ~]# systemctl disable NetworkManager
[root@osp9 ~]#
```

- Then stop our Network Manager

systemctl stop NetworkManager

```
[root@osp9 ~]# systemctl stop NetworkManager
[root@osp9 ~]#
```

- after these steps enable Network

systemctl enable network

- Then start Network

systemctl enable network

- Install the OpenStack repository on CentOS 7

yum install -y centos-release-openstack-queens

```
[root@osp9 ~]# yum install -y centos-release-openstack-queens
Loaded plugins: fastestmirror, langpacks
Repository rdo-trunk-queens-tested is listed more than once in the configuration
Loading mirror speeds from cached hostfile
 * base: centos.mirror.net.in
 * extras: centos.mirror.net.in
 * updates: mirror.vbctv.in
Package centos-release-openstack-queens-1-1.el7.centos.x86_64 already installed and lat
est version
Nothing to do
[root@osp9 ~]#
```

- Make repository enable

  # yum-config-manager –enable openstack-queens

```
[root@osp9 ~]# yum-config-manager --enable openstack-queeens
Loaded plugins: fastestmirror, langpacks
Repository rdo-trunk-queens-tested is listed more than once in the configuration
[root@osp9 ~]#
```

- Then check your current packages is latest or not, if not then update your packages.

  #yum update

- Install your packstack

yum install -y openstack-packstack

```
[root@osp9 ~]# yum install -y openstack-packstack
Loaded plugins: fastestmirror, langpacks
Repository rdo-trunk-queens-tested is listed more than once in the configuration
Loading mirror speeds from cached hostfile
 * base: centos.mirror.net.in
 * extras: centos.mirror.net.in
 * updates: mirror.vbctv.in
Package 1:openstack-packstack-12.0.0-2.el7.noarch already installed and latest version
Nothing to do
[root@osp9 ~]#
```

- Last step, Deployment of single node openstack

yum packstack –allinone

6. MARIADB

MariaDB is one of the most popular database servers in the world. It is originally developers of MySQL and guaranteed to stay open source. It is structured information in a wide array of applications. It is an enhanced, drop in replacement for MySQL. MariaDB is fast, scalable and robust with a rich ecosystem of storage engines, plugins and many other tools make it very versatile for a wide variety of use cases. It provides an SQL interface for accessing data.

MariaDB Config. Using CentOS 7 in VMware Workstation pro 14.

We are going to install and configure MariaDB server.

- Install MariaDB packages
  #Yum install -y mariaDB Server

- Enable the service and start the service
  # systemctl enable mariadb
  # systemctl start mariadb

- Allow incoming connections to the standard MySQL TCP port 3306.
  # firewall-cmd –permanent –add-service=mysql
  # firewall-cmd –reload

- Secure MariaDB, set password for the database root user, disable remote root access and remove the test database and my anonymous users.
  # mysql_secure_installation

- Restart the service
  # systemctl restart mariadb

- Test root login by executing below command
  # mysql -uroot –p(password)

- Execute a simple SQL query
  MaraiDB [(none)]>select@@version;

Creating a New Database and a New User

- Login as a MariaDB root user.
- List existing users, create a new database test2
- Grant all privileges on the database test2 to the database user dbuser1,
- Flush privileges and show grants for the newly created user.
  # mysql -uroot -p(password)

- MariaDB[(none)]> SELECT user,hostFROM mysql.user;
- MariaDB[(none)]> CREATE DATABASE test1;
- MariaDB[(none)]> SET old_passwords= 0;
- MariaDB[(none)]> CREATE USER 'dbuser1'@'localhost' IDENTIFIED BY "password";
- MariaDB[(none)]> GRANT ALL PRIVILEGES ON test1.* TO 'dbuser1'@'localhost' IDENTIFIED BY "password";
- MariaDB[(none)]> FLUSH PRIVILEGES;
- MariaDB[(none)]> SHOW GRANTS FOR 'dbuser1'@'localhost';
- MariaDB[(none)]> SELECT user,host,Grant_priv,Super_privFROM mysql.user;
- MariaDB[(none)]> exit

Performing simple Queries against a database

- MariaDB[(none)]> use test1;
- MariaDB[test1]> create table services (id INT(10) unsigned, name VARCHAR(20), version INT(10));
- MariaDB[test1]> show tables;
- MariaDB[test1]> describe services;
- MariaDB[test1]> insert into services (id, name, version) values (1, "apache", "2");
- MariaDB[test1]> insert into services (id, name, version) values (2, "samba", "4");
- MariaDB[test1]> select * from services;
- MariaDB[test1]> delete from services where id=1;
- MariaDB[test1]> select * from services;
- MariaDB[test1]> exit

7. FILE TRANSFER PROTOCOL (FTP) CONFIGURATION

- Install vsftpd
  # yum install vsftpd ftp -y

- Configure vsftpd
  # vi /etc/vsftpd/vsftpd.conf

  Find the following lines and make the changes as shown below.

```
## Disable anonymous login ##

anonymous_enable=NO

## Uncomment ##

ascii_upload_enable=YES

ascii_download_enable=YES

## Uncomment - Enter your Welcome message - This is optional ##

ftpd_banner=Welcome to UNIXMEN FTP service.

## Add at the end of this  file ##

use_localtime=YES
```

- Enable and Start the vsftpd service
  # systemctl enable vsftpd
  # systemctl start vsftpd

- Firewall configuration
  # firewall-cmd –permanent –add-port=21/tcp

firewall-cmd –permanent –add-service=ftp
firewall-cmd –reload

- SELinux configuration
  #setsebool -P ftp_home_dir on

- Create FTP users
  # useradd admin
  # passwd admin

- Connecting to FTP server
  # ftp 192.168.137.2
  Enter the ftp username and password.
  Sample output:

```
Connected to 192.168.1.101 (192.168.1.101).

220 Welcome to UNIXMEN FTP service.

Name (192.168.1.101:root): sk

331 Please specify the password.

Password:

230 Login successful.

Remote system type is UNIX.

Using binary mode to transfer files.

ftp>
```

- Client-side configuration
  # ftp 192.168.169.2

Sample Output:

```
Connected to 192.168.1.101.

220 Welcome to UNIXMEN FTP service.

Name (192.168.1.101:sk): sk

331 Please specify the password.

Password:

230 Login successful.

Remote system type is UNIX.

Using binary mode to transfer files.

ftp>
```

Our FTP server is working.