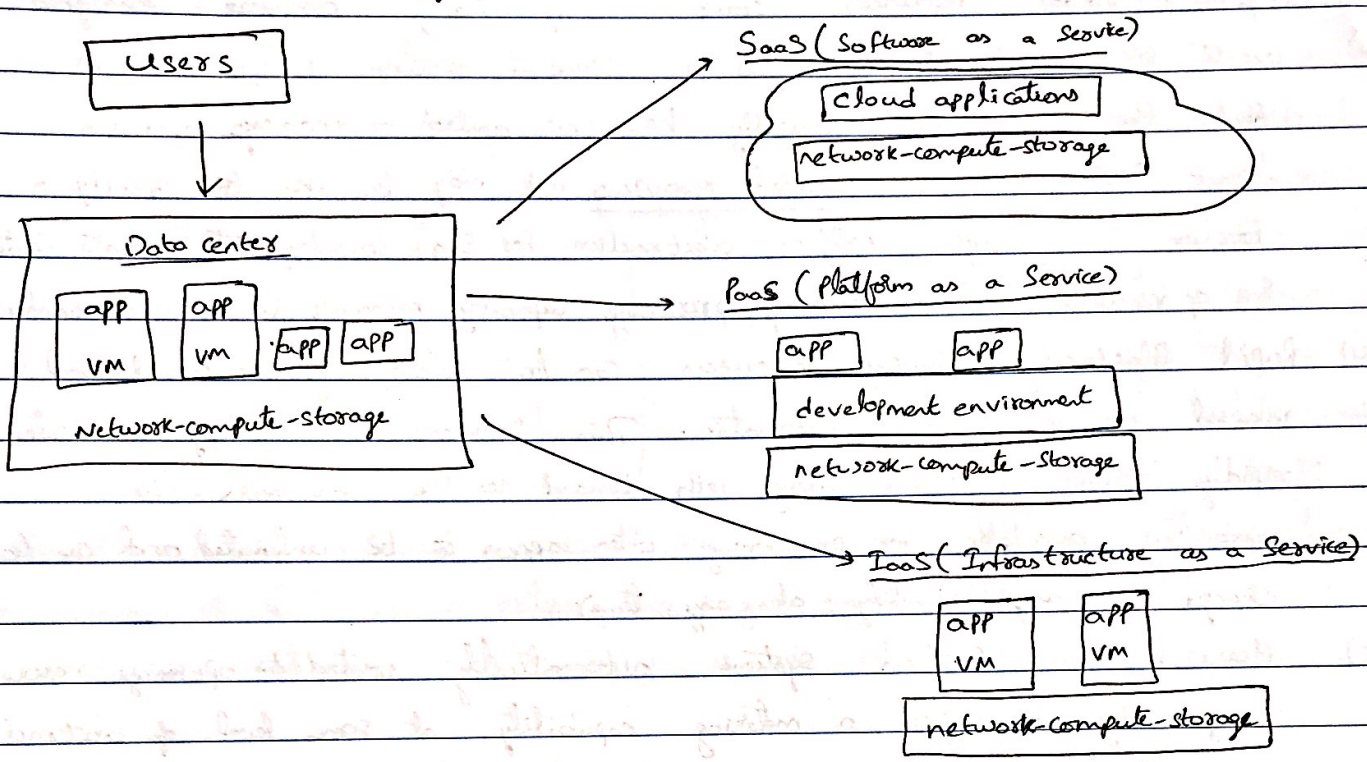
OPENSTACK

Introduction



→ SaaS, IaaS & PaaS are 3 delivery models of Cloud Computing.

What is Cloud Computing?

↳ It is a type of internet based computing that provides shared computer processing resources and data to any type of consumer device. It is a model for enabling ubiquitous on demand access to a shared pool of configurable computing resources.

These computing resources could be computer networks, servers, storage, applications and services which can be rapidly provisioned & released with minimal management efforts

Essential Characteristics of Cloud Computing:

1) On-Demand self-service (A consumer can provision computing capabilities such as server network storage or an application by himself without requiring human interaction with service provider

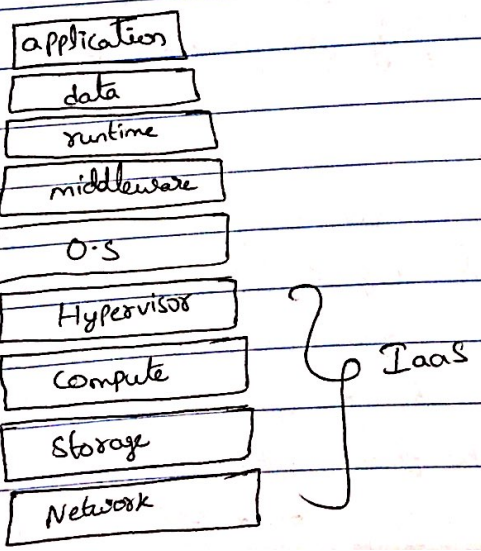
2) ~~Broad~~ Broad Network Access (This emphasizes the fact that services should be available over the network and can be accessed through standard mechanisms. Any client platforms like phones, laptops or high end servers could be used for that.

3) Resource Pooling (This means the providers computing resources are pooled to serve multiple consumers using a multi-tenant model. Different physical & virtual resources could be dynamically assigned & reassigned acc. to consumer demand. There is a sense of location independance in that the consumer generally has no control or knowledge over the exact location of the provided resources but may be able to specify a location at a higher level of abstraction. for E.g Country, state or Data Center E.g of resources include storage, processing capacity, memory & Network bandwidth)

4) Rapid Elasticity (Meaning Resources can be elastically provisioned and released in some cases automatic. This provides the ability to scale rapidly outward & inward along with demand to the consumer. The capabilities available for provisioning often appear to be unlimited and can be changed in any quantity at any time)

5) Measured Service (Cloud systems automatically control & optimize resource usage by leveraging a metering capability at some level of abstraction appropriate to that type of service. Depending on the service metered, resources could be storage capacity, processing capacity, bandwidth & active user accounts. Resources usage can be monitored controlled & reported, providing transparency for both the provider & consumer of the utilities service in the end

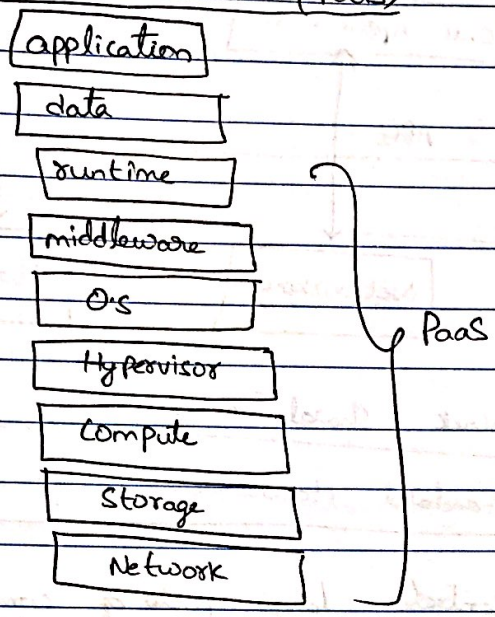
=> Infrastructure As A Service (IaaS)



Openstack clouds are good example of IaaS

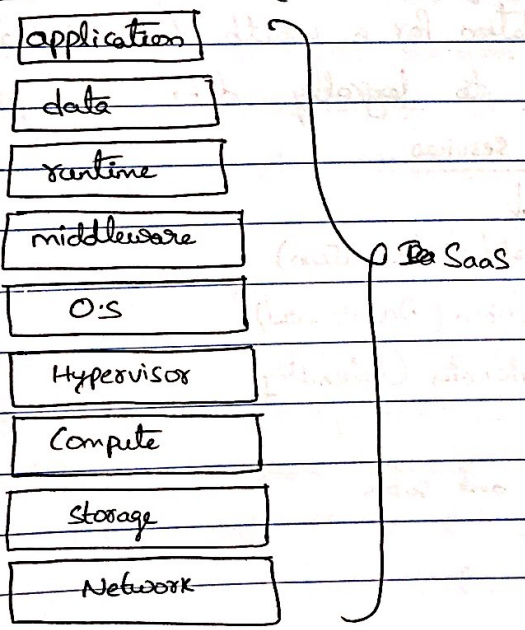
IaaS abstracts these bottom four layers for us, so we are free we don't have to to choose O·S of our own choice, as well as middleware etc.. All we do is determine our needs & say I need this amount of storage, CPU & RAM.

⇒ Platform As A Service (PaaS)

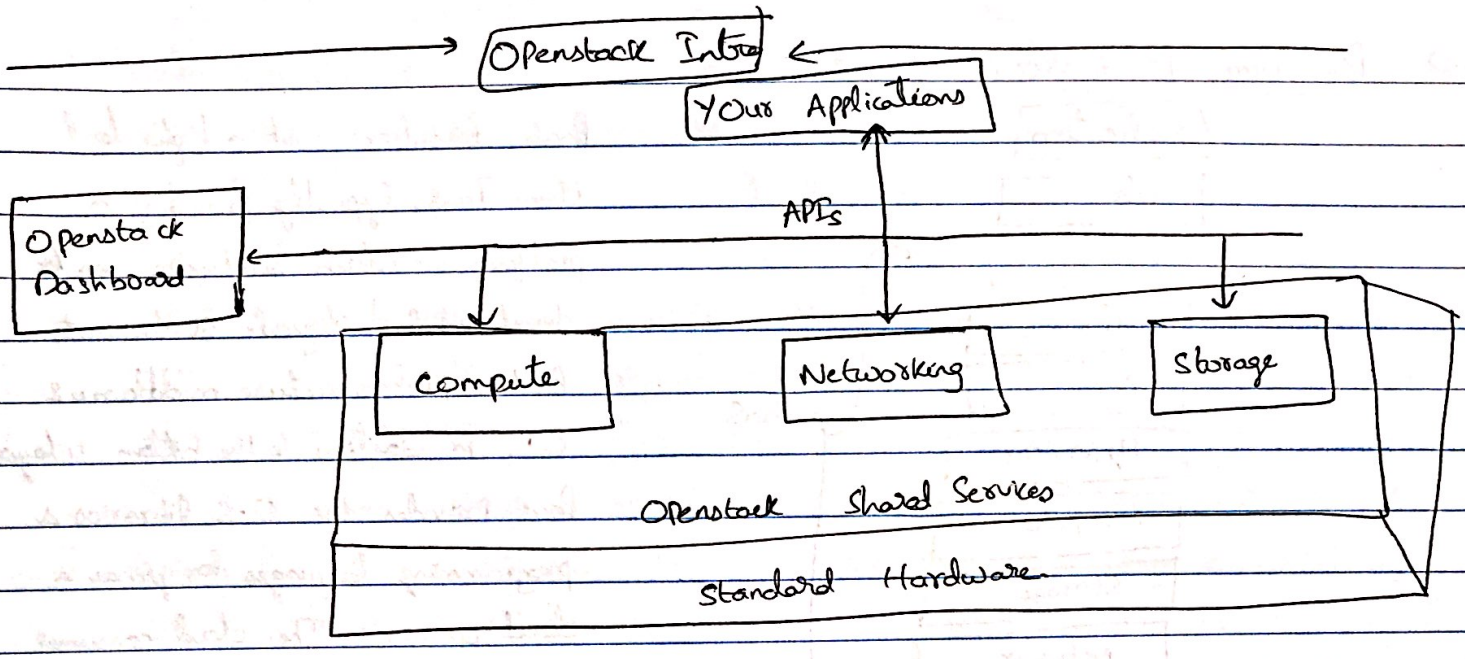


PaaS functions at a higher level than IaaS, typically providing a platform on which software can be developed & deployed. It abstracts Runtime, infrastructure middleware & O·S in addition to the bottom 4 layer PaaS provides the O·S libraries & programming languages for you as a cloud consumer. The cloud consumer provides the application to be deployed on the instances & doesn't managing the underlying infrastructure. An example of PaaS is an eclipse Java programming platform provided with no downloads required. We don't deal with setting up the environment.

⇒ Software As A Service (SaaS)



SaaS is the delivery model where the cloud provider gives access to an application.



→ OpenStack is an IaaS that controls large pools of compute, storage and networking resources throughout a data center.

→ All resources are managed through a dashboard that gives administrators control while empowering their users to provision resources through a web interface

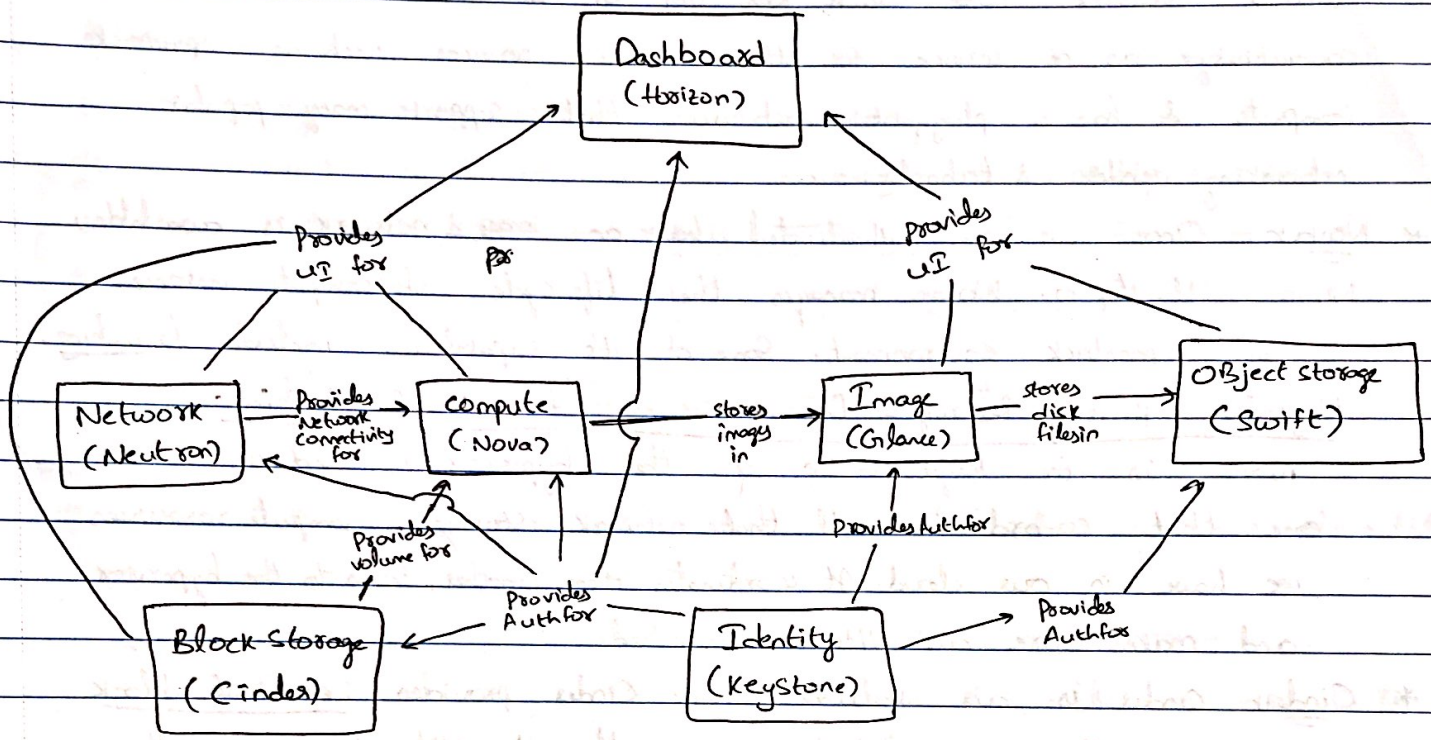
→ Openstack has a modular architecture with various code names for its component projects. It is a powerful & highly configurable integration engine with a set of core projects that are a foundation for a wealth of applications.

→ The OpenStack Foundation has decided to logically define the projects into 2 → Core Services & Optional Services

Core Services:
- Nova (compute)
- Cinder (Block storage)
- KeyStone (Identity)
- Swift (Object storage)
- Neutron (Networking)
- Glance (Image)

Optional Services:
- Heat (Orchestration)
- Horizon (Dashboard)
- Ceilometer (Telemetry)
- ⋮
- and so on.

⟹ OVERVIEW (LOW-LEVEL)↓



* Keystone: The identity service, this is where we'll get authenticated and authorized within openstack. Everyone has an identity and needs to be authenticated with one another. We are talking about end users as well as these components projects together.

The 2nd imp function of Keystone is it where all the services get registered to, as the central registration point. It provides a catalog to services & users so they know how to reach openstack services.

* Glance: Image Management. It stores & retrieves virtual machine disk images. Openstack compute makes use of this during instance provisioning when a VM starts. It must have an image behind it. We need to pre-build the images and load them into glance beforehand so than when VM starts it goes out to glance & pulls a copy of the image from there inside instead of us having to go through the whole installation process each time it's pre-installed for us. & ready

* Neutron: Neutron is openstack networking and it will create that virtual network and attach the VM to it. Neutron enables network connectivity as a service for other openstack services such as openstack Compute & has a pluggable architecture that supports many popular networking vendors & technologies.

* Nova: Once we are authenticated, have an image & network is available, Nova will step in. Nova manages the life-cycle of compute instances in an openstack environment. Some of it's capabilities include launching, Migrating, Pausing, Resizing & Decommissioning of Virtual Machines.
Nova is the layer on top of the hypervisor is and it is the one that co-ordinates all that network storage & compute resources we have in our cloud. It coordinates them together, talks to the hypervisor and makes sure our VMs are launched.

* Cinder: Cinder is our volume service. Cinder provides persistent block storages to the running instances. Normally, the VMs boot up with ephemeral volumes. That means when we terminate the VM the data within the VM is lost. If we need to keep the data when it's terminated attach a persistent block device to those volumes so that VM can write the info. that needs to persist on that volume. So, when that VM gets terminated can write the info. that needs to persist on that volume. When the VM gets terminated we're able to reattach that block volume device to another instance. This provides the ability to continue using the info. that we have have and be terminated instance.
This isn't shared storage, it's block storage. Meaning there is a one to one relationship b/w the volumes & instance.

* Swift: This provides the Object Storage. Swift helps us to store & retrieve simple objects like an MP4 video file. It uses an Https based API where we pass and objects & metadata with regular http GET & PUT commands.

* Horizon: This provides a web-based self-service portal to interact with underlying openstack services such as launching an instance, assigning IP addresses, configuring access controls & so on..

* Note: We can check the state of each of the openstack services to all the systemd services running on our node by running below command

$ ↳ cat /etc/systemd/system/multi-user.target.wants/ && ls *.service

↳ systemctl status <my-service>

* Note: Horizon & Dashboard are sometimes used interchangably, but there is some basic difference. Horizon is the underlying framework and the dashboard is the web interface built on top of that.

Horizon

→ The password for admin account can be found on keystonerc_admin file.

Openstack CLI

→ First, for using the openstack commands we have to authenticate. So, we source a credentials file. The credentials file is rc file. The file name of RC file may be different with different openstack distros out there.

E.g. of Sovereign: source /root/openstack-configs/openrc.

Now we can try any openstack commands ⇒ E.g. openstack server list.

Note 2 → Unified Command Line

↳ Previously each project like nova, neutron, glance etc.. had its own command line tool. This has different capabilities and formatting which created bad user experience.

↳ Then, came unified command line:

⇒ nova boot → openstack server create

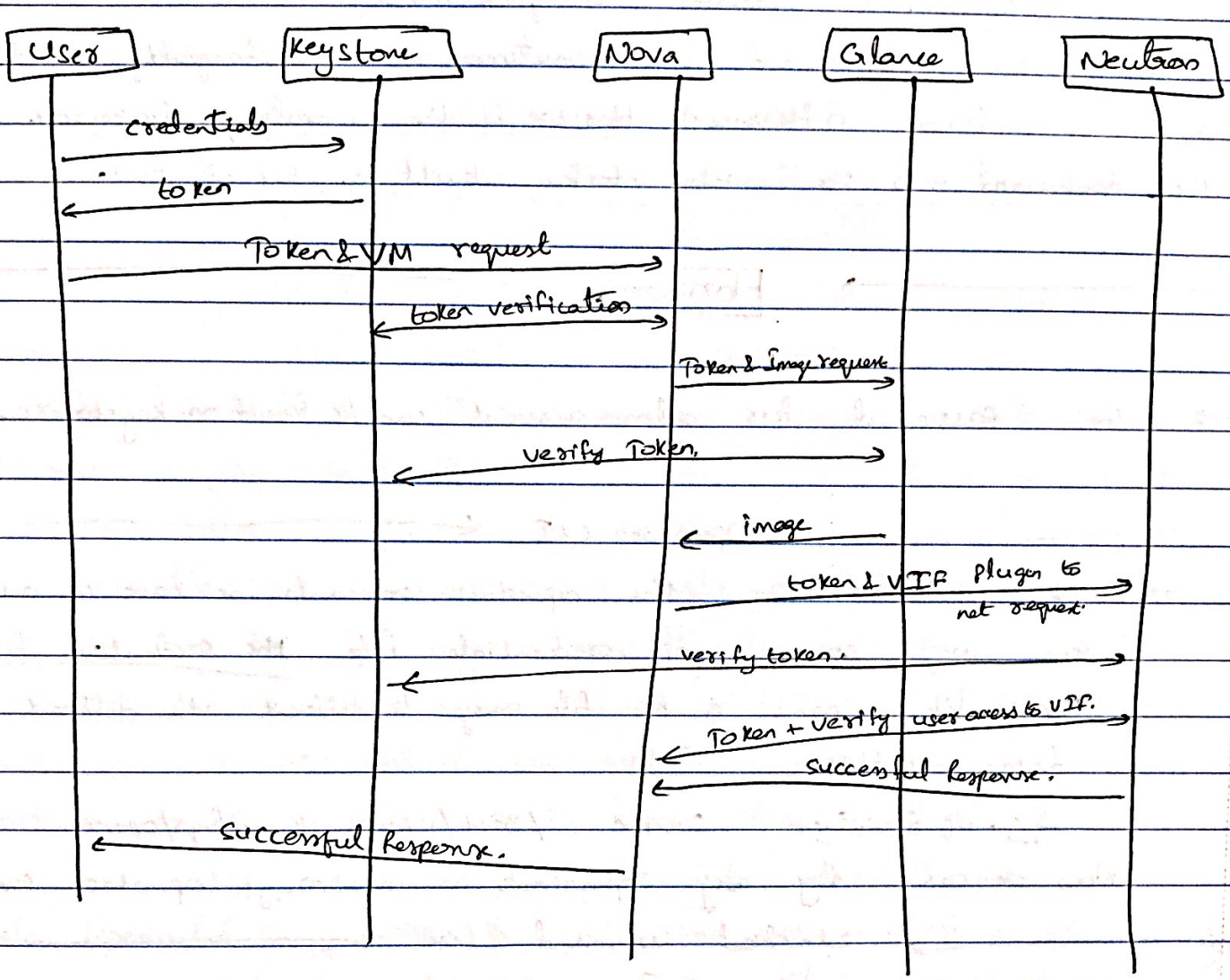
⇒ neutron net-create → openstack network create

⇒ glance image-list → openstack image list.

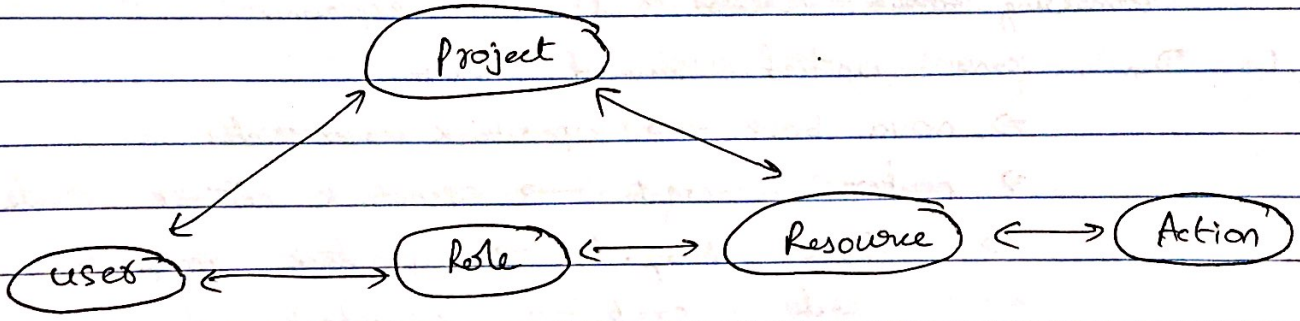
⇒ cinder create → openstack volume create.

→ Key-stone ←

⇒ Authentication Process flow:



⇒ Policies & Authorisation



Summary:- 1) Common authentication System
2) Central catalog of Services & endpoints
3) Supports LDAP, AD, MySQL
4) Provides a token for subsequent auth. requests

=> Keystone CLI Most used Commands:-

1) source /root/openstack-configs/openrc
(Everytime we should run this before using openstack commands to get authenticated)
2) openstack endpoint list (will list all the endpoints of openstack)
3) openstack endpoint show <UUID> (will show detailed info of endpoint)
4) openstack catalog list (This will list out the endpoints and the IP address which they use to reach out to other services)

Note:- a) We notice that there are services like cinderv1, cinderv2 & v3
This is the API versions changes that are regularly introduced to the API for adding new functionality & sometimes fixing a bug.
* Sometimes the API itself needs to change to incorporate new methods which is not backward compatible. This is where they implement a new version of the API & keep the previous one for some time to remain compatible with others & give community some time to implement the new API version.

5) openstack project create <project_name> (creates new project)
6) openstack project show <project_name/id> (shows detailed info of project)
7) openstack project set --description "description text" <project_name>
(After project is created, we can set the description of project with like above command)
8) openstack user create --project <project_name> --password-prompt <user_name>
(creating users on for specific project with password prompt)
9) openstack role list (Lists out the role list)
10) openstack role add --project <project_name> --user <user_name> _member_
(for creating assigning role to a user => _member_ → for member admin → for admin privileges)
11) openstack command list | grep openstack.identity -A 40
(Lists out all the openstacks commands related to identity)

GLANCE

1) Used to store cloud images & snapshots
2) Has client-server architecture that provides a rest API to the users through which requests to the server can be performed.
3) Uses swift or other as object storage for back-end.

⇒ Some of the supported Image types by Glance:
1) Raw
2) Machine (kernel/ramdisk outside of image, also known as AMI)
3) VHD (Hyper-V)
4) VDI (Virtualbox)
5) OVF (VMWare, others)
6) Also supports Docker Containers & Bare-Metal Servers as well.

⇒ Managing Glance from CLI:
1) openstack command list | grep openstack.image -A 20
(Lists out all the commands related to image)
2) openstack image list (Lists out all the images available in our Glance)
3) openstack image create --min-disk <X GB> --private --disk-format <disk_format> --file <image_file> <image_name>
(The above command is for creating images in Glance.
--min-disk => <X GB> => (optional) (ensures minimum disk size needed to boot image, in gigabytes)
--private => (optional) (ensures image is only visible to the users in the project)
4) openstack image show <image_name_or_ID> (shows detailed info of images)
5) openstack image delete <image_name_or_ID> (Deletes the image from Glance repository)
6) openstack image save <image_name_or_ID> (Saves an existing image in a glance repository to a local file)
7) openstack image add project <image_name_or_ID> } → Adds/removes the image from the project not from the glance repository)
8) openstack image remove project <image_name_or_ID> }
9) ~~openstack image~~ openstack image set <image_name_or_ID> }
" " unset " " } → for setting/unsetting image properties & tags.

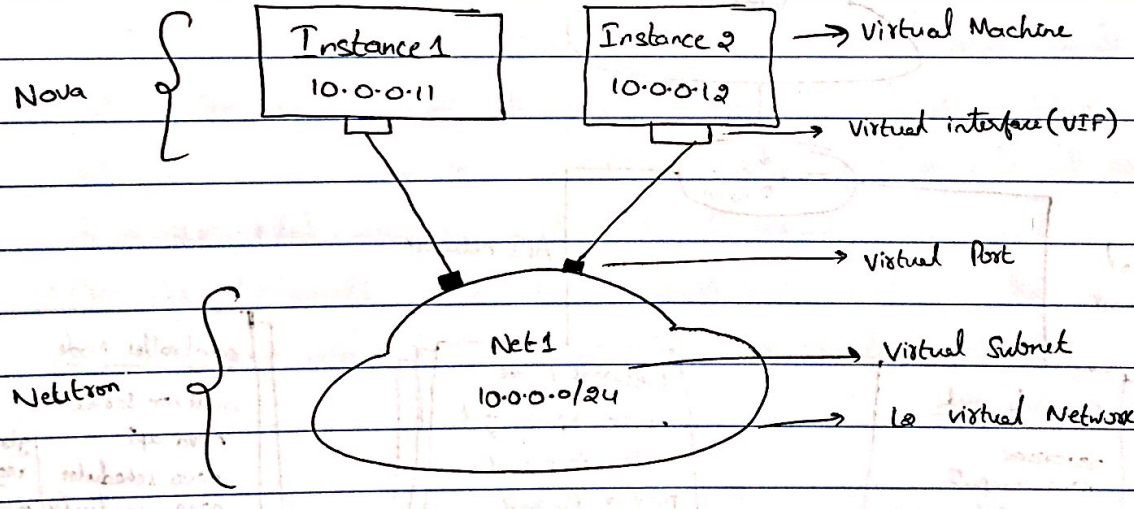
→ NEUTRON ←

→ Network connectivity as a service
→ Network, subnet & port abstractions
→ Plugins support many technologies
→ Has Modular Architecture which we could deploy either in a centralized or distributed way depending on our needs.
→ The service works by allowing

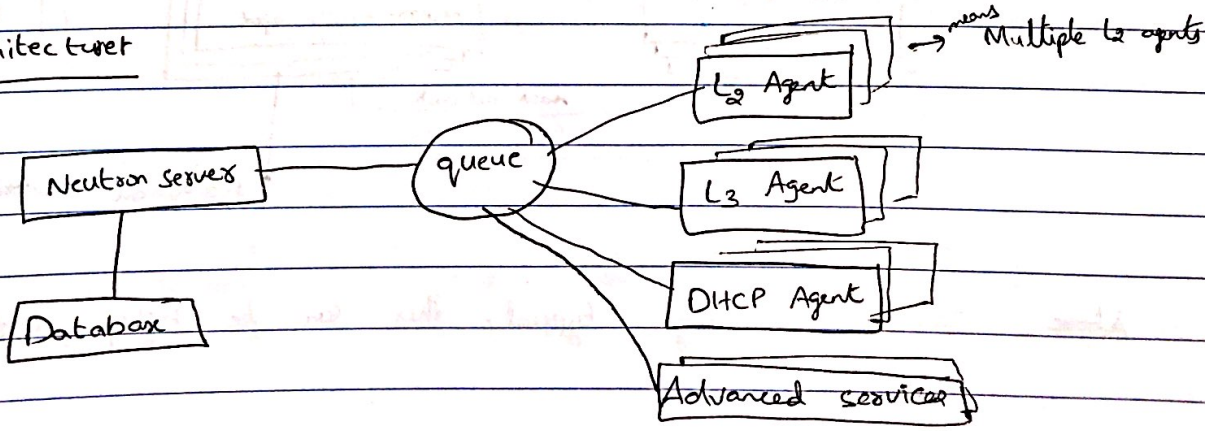
⇒ Benefits of Neutron

1) Rich Topologies
2) Enables advanced services like Load Balancing, VPN, Firewall and so on.

⇒ Base Terminology & Abstractions



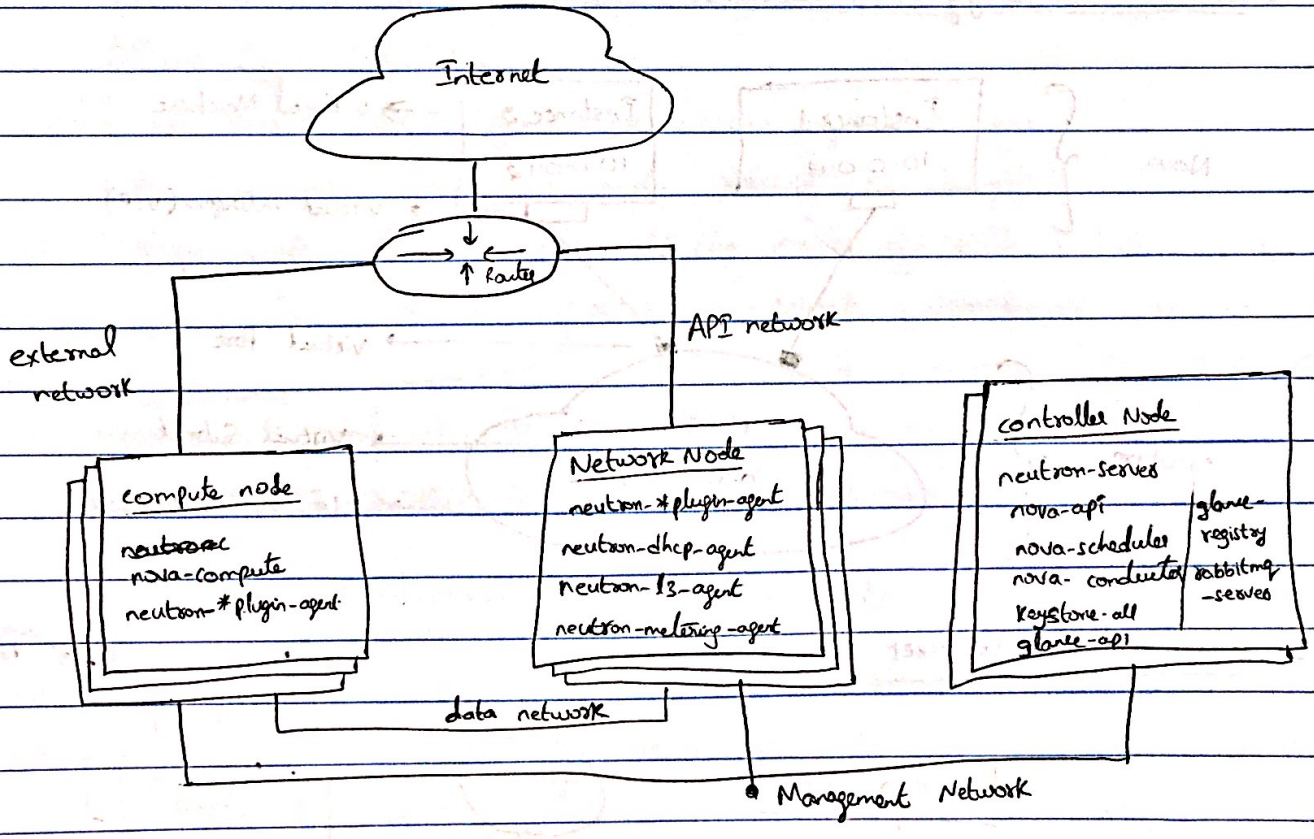
⇒ Neutron Architecture



↳ The Message queue is used for exchanging messages with other neutron agents, typically in a deployment we will find an L2 → Layer-2 Agents. This * L2 agent is responsible for wiring up ports and the devices and connecting them into shared broadcast domain.

* Another one will be DHCP Agent, this allows us to configure the instance IP address. (There is also another way which is config-drive).
* L3 agents is responsible for providing connectivity among different networks. If our instances have connectivity to the external networks this is where their traffic passes through.
* Typically the L2 agents reside on the hypervisors of the compute node.
* Advanced services are for doing a load balancing, firewalling or VPN gateway.

> Which Neutron Component Resides where?

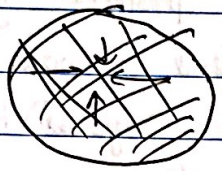


Above scenario is only typical, This can be highly customizable.

⇒ Provider & Project Networks

↳ Multi-Tenancy

→ Virtual isolated networks can be created inside of openstack projects. These networks only have routes to the outside world if we create them. We need to design our network acc. to needs of our environment.



↳ In Neutron, Networks can be divided into 2 types.

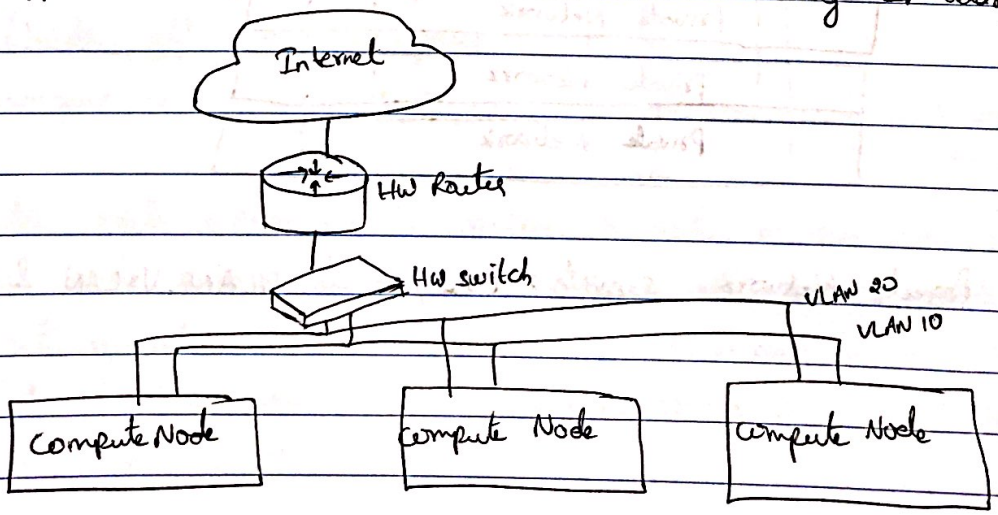
1) Project Networks 2) Provider Networks.

1) Project Networks are created by normal users & details about how they are physically realized are hidden from these users

2) provider networks are created by administrators by specifying the details of how the network is physically realized. And usually it matches some existing network in data center. This is most commonly used to give projects direct access to a public network that can be used to reach internet.

Supports: Flat Network, VLAN, VXLAN, GRE

*Provider Networks only handle layer-2 connectivity for instances thus lacking supports for features such as Routers & Floating IP addresses

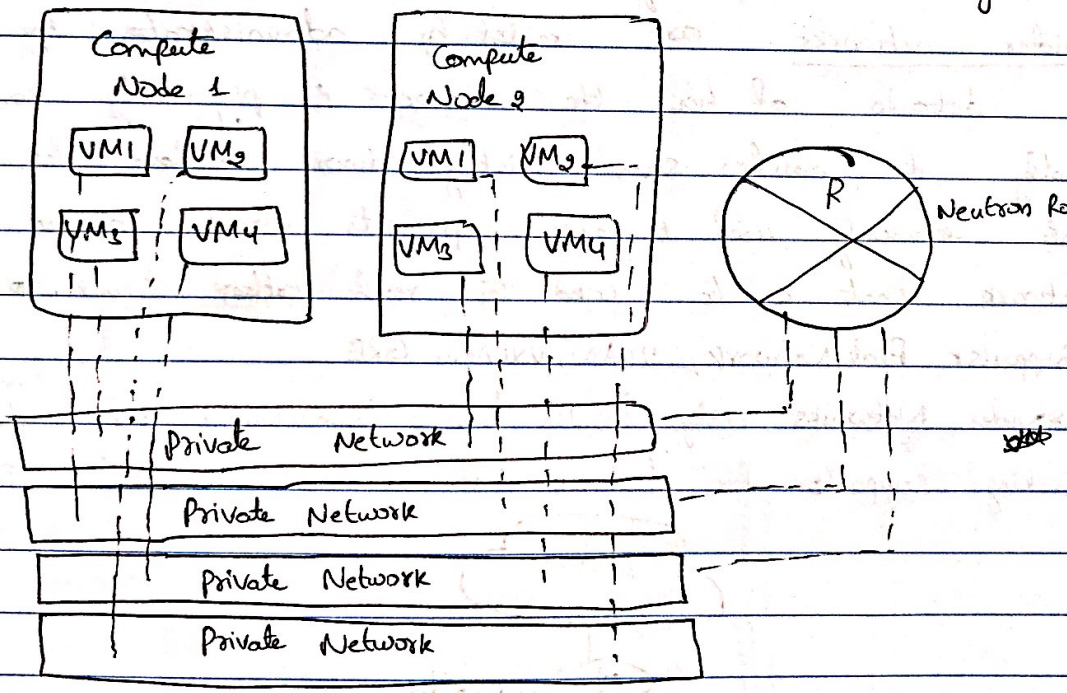


→ Project Networks: Project & Tenant could be interchanged & mean the same thing. The primary difference b/w project networks & provider networks revolves around who provisions them.

Provider networks are created by the administrator on behalf of cloud tenants & could be used by one or more tenants on the other hand. Project network are created by project users to be used by their instances inside the project. Typically, provider networks are directly associated with a physical network like specific virtual LAN in Data Centre.

Note 2: Provider Networks Rely on the physical Network Infrastructure to provide default gateway or 1st hop routing services.

Project/Tenant Networks rely on Neutron Routers to fulfill this Role. Neutron Routers must attach their upstream interfaces to provider networks, designated as external in order to connect to the physical Network.



Project Network supports: Local, Flats, VLAN, VxLAN & GRE

⇒ Neutron - CLI commands

1) openstack network agent list (Lists all the network agents)
2) systemctl status neutron-server (To see the status of Neutron Server)
3) ovs-vsctl show (shows the ovs bridges)
   ↳ This will give the o/p of list
   a) br-int (Integration Bridge) → Bridges the traffic b/w the instance and the tunnel & external bridges
   b) br-tun (Tunnel Bridge) → for creating VXLAN & GRE Tunnels b/w nodes
   c) br-x (External Bridge) → for connecting to the external Physical Network

4) openstack network create -h
   --share ⇒ if we want to share network b/w projects
   --project <project> ⇒ To assign a specific project as admin
   --project-domain <project-domain> ⇒ This can be used in case there is collision b/w project name exists
   ~~--availability-zone-hint <availability-zone>~~

5) openstack subnet create <subnet_name> --subnet-name <IP_subnet>
   --dns-nameserver <DNS_IP> --network <network_name>
   (once we created network, Next is creating subnet & assigning it to network created)

6) ip netns (Lists all the namespaces we have)
7) ip netns <namespace_name> exec ip show (Lists all the interfaces & ip addresses we have under namespaces)

8) openstack router create <name> (For creating the router for our subnet)
9) openstack router add subnet R2 <subnet_name> (To add subnet to router)
10) neutron router-gateway-set R2 external_network (Next is assigning gateway for router)
    (Here we attached the external public network to our router so that the router can route traffic to & from that network)

ingress → incoming traffic
egress → outgoing traffic

11) openstack server create --image <centos> --flavor 1 --nic net-id=<UUID> inst1
(we launched an instance to our newly created network)
Before connecting to the instance we need to change the security group rules applied to it so that it allows for incoming traffic

12) By default security group rules drop all incoming traffic we need to allow ICMP & ssh traffic to the instance. so for that we need to get ID of our security group that's apply it to our instance.

12) openstack security group list
13) openstack project list

14) So after getting ID we need to use below command to create rule for allowing
a) ICMP traffic

openstack security group rule create --src-ip 0.0.0.0/0 --protocol icmp --ingress <sec-group-id>

b) SSH traffic

openstack security group rule create --src-ip 0.0.0.0/0 --dst-port 22 --protocol tcp --ingress <sec_group_Id>

15) Let's connect to instance from name space.

ip netns exec <router_namespace> ping <instance_ip>
ip netns exec <router_namespace> ssh <instance_ip>

16) Instances require external connectivity most of the time. If it's serving the public internet or any other network that is outside of openstack it needs to be reachable from external networks The way to do that is via floating IPs
So, let's assign a floating IP to instance & test it's connectivity

a) openstack subnet list → This will give subnet ID & Network ID
b) openstack floating ip create --subnet <subnet_ID> <network_ID>
After it is created lets assign it to the instance.
c) openstack server add floating ip <instance_name> <floating_ip>

17) openstack ip availability list --project <project_name>
(This gives info about IP availability inside a project)

18) openstack command list | grep openstack.network -A 70

=>(address scope) commands are for creating listing & managing address scope's which prevents address overlapping

=> ('address floating ip) commands are for managing floating IPs we assigned to instances for a one to one address translation.

=> (network rbac) commands sets a role based access control RBAC policy for network resources. It enables both operators & users to grant access to network resources for specific projects

=> (network segment) commands are for further segmentation of Networks. Commands
A Network Segment is an isolated layer to segment within a network & network may contain multiple network segments. layer 2 connectivity b/w network segments within a network depends on configuration

=> (port) commands. A port is a connection point for attaching a single device such as the Network interface card of a server to a network. The port also describe the associated network configuration such as the MAC & IP addresses to be used on that port. with

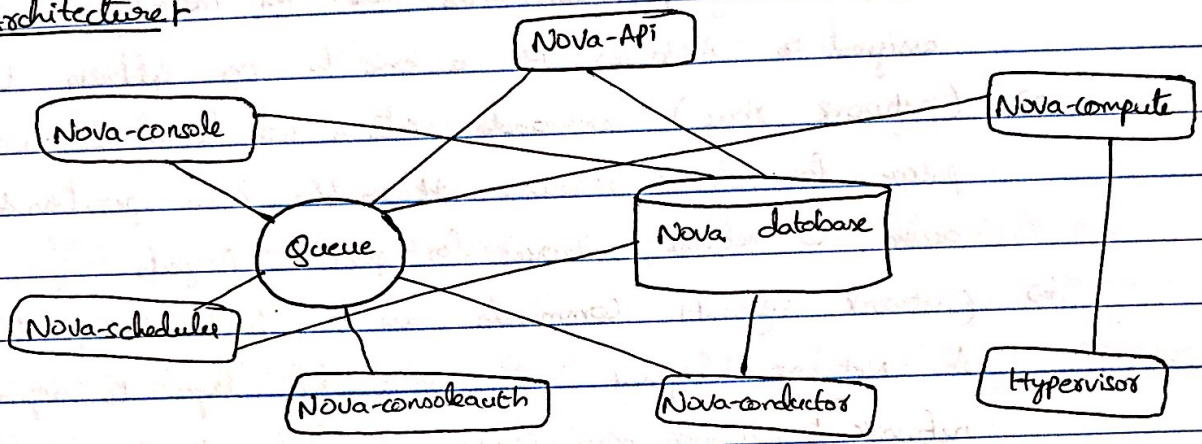
=> Routes commands are for managing our virtual Routes life-cycle.

=> (Security group commands) A security group acts as a virtual firewall for servers & other resources on a network. It is a container for Security group rules which specify the network access rules.

NOVA

→ Nova is not a hypervisor but instead it manages the hypervisor. Nova is the layer on top of it. It is is responsible for interfacing with multiple hypervisor and managing the instance lifecycle.

⇒ Nova Architecture:



Nova-api: This API service exposes restful interface for other components or users to talk with. It supports openstack compute API & a special admin API for privileged users to perform administrative actions. It also initiates most of the orchestration activities such as running an instance as well as enforces some policies like quota checks

Queue: All our communication within the project is via an RPC message queue. It is like a central hub for passing messages b/w daemons. This is usually implemented with RabbitMQ today.

Nova-conductor: This is responsible for taking the instance build request or requesting the scheduler to find on which node it's going to launch it on. And also for interacting with the database & compute nodes. The reason we have the conductor is for security. There is no need to have database credentials on every single compute node. Instead of that the compute nodes talk to database through conductor

Nova-Scheduler: It takes a virtual Machine instance requests from the queue & determine on which compute node it should run.

Nova-Compute: This compute agent, in the case of KVM Hypervisor it is going to be talking directly to libvirt on that machine or in case of something like VCentre driver it's talking to VSensors through its own native APIs.

The Nova compute process is primarily a worker daemon that creates & terminates VM instances via Hypervisor's APIs.

Regardless of the complexity of the process by which it handles the basics are simp Accepts actions from queue and then perform a series of system commands like launching a KVM instance while updating the state in the database.

Nova-database: The SQL database stores most of the build time & run-time state for a cloud infrastructure. This includes the instance types that are available for use, instances in use, Networks available & projects.

→ Finally, Nova interacts with many other openstack services like Keystone for authentication Glance for images & Horizon for web interface.

⇒ LAUNCHING AN INSTANCE:

↳ Prerequisites for launching an instance:

1) An Image
2) A Network
3) A Flavor (RAM, vCPU, storage)

→ Instance Creation

↳ Openstack server create command with minimal arguments:

$ openstack server create --image <image> --flavor <flavor> --nic net-id=<net-id> instance-name

→ Flavor determine the instance size
→ Network determines where to attach the instance to
→ Image determines the OS image used to boot the instance.

→ Image selection:

openstack image list

openstack image show <uuid>

→ Flavor selection:

By default we have some flavors & also have custom flavors.

openstack flavor list

→ Network selection:

openstack network list

→ Finally for launching an instance →

openstack server create --flavor 1 --image <imagename> --nic net-id=<net-id> <instance_name>

when we run this command we get o/p like this

task_state: scheduling
vm_state: building

} it means database entry has been created at this point for requested instance and the process is still continuing

⇒ Segregation of Compute Resources:

→ Providing logical groupings:

↳ Data Center, Geographical Region, Power source, Rack, Network Resources

→ Differentiate specific H/w on compute nodes:

↳ GPU cards, Fast NICs, storage devices, SSDs

Different ways of segregation:

1) Region:

2) Host Aggregate:

a) Logical Grouping of compute nodes based on metadata

b) Typically metadata describes capabilities of nodes:

↳ SSD hard disks ↳ 40G NICs ↳ GPU cards.

c) A compute node can be in multiple host aggregates:

↳ Hosts with dual 40G NICs & SSD storage.

→ Host aggregate are not chosen explicitly by the user, but the user can implicitly target a host aggregate. This differs from explicitly specifying the region.

As a user, we won't specify which type of host aggregate we are targeting. We wouldn't say I want an aggregate with this name for an instance. typically. Instead the admin creates a flavor, creates some metadata in parallel & matches that to the aggregate mediated.

$ openstack aggregate create nodes-with-SSD

$ openstack aggregate set --property SSD=true nodes_with_SSD

$ openstack aggregate add host nodes_with_ssd host1

$ openstack flavor set --property SSD=true m1-large

User selects flavor when requesting instance.

Scheduler chooses host matching flavor extra specs with host aggregate metadata.

3) Availability zones:

These only differ slightly compared to host aggregates. ~~availability~~ Like Host aggregates they are logical Grouping of hosts based on factors like:

a) Geo-Locations (Country, city, Data Center, Rack etc.)

b) Network Layout

c) Power source

We could use them to create different failure domains within our compute nodes. Unlike host aggregates they are explicitly user targetable. This means I can explicitly state from dashboard or CLI that I want an instance in a specific availability zone. If I don't specify, the instance will be launched from default availability zone.

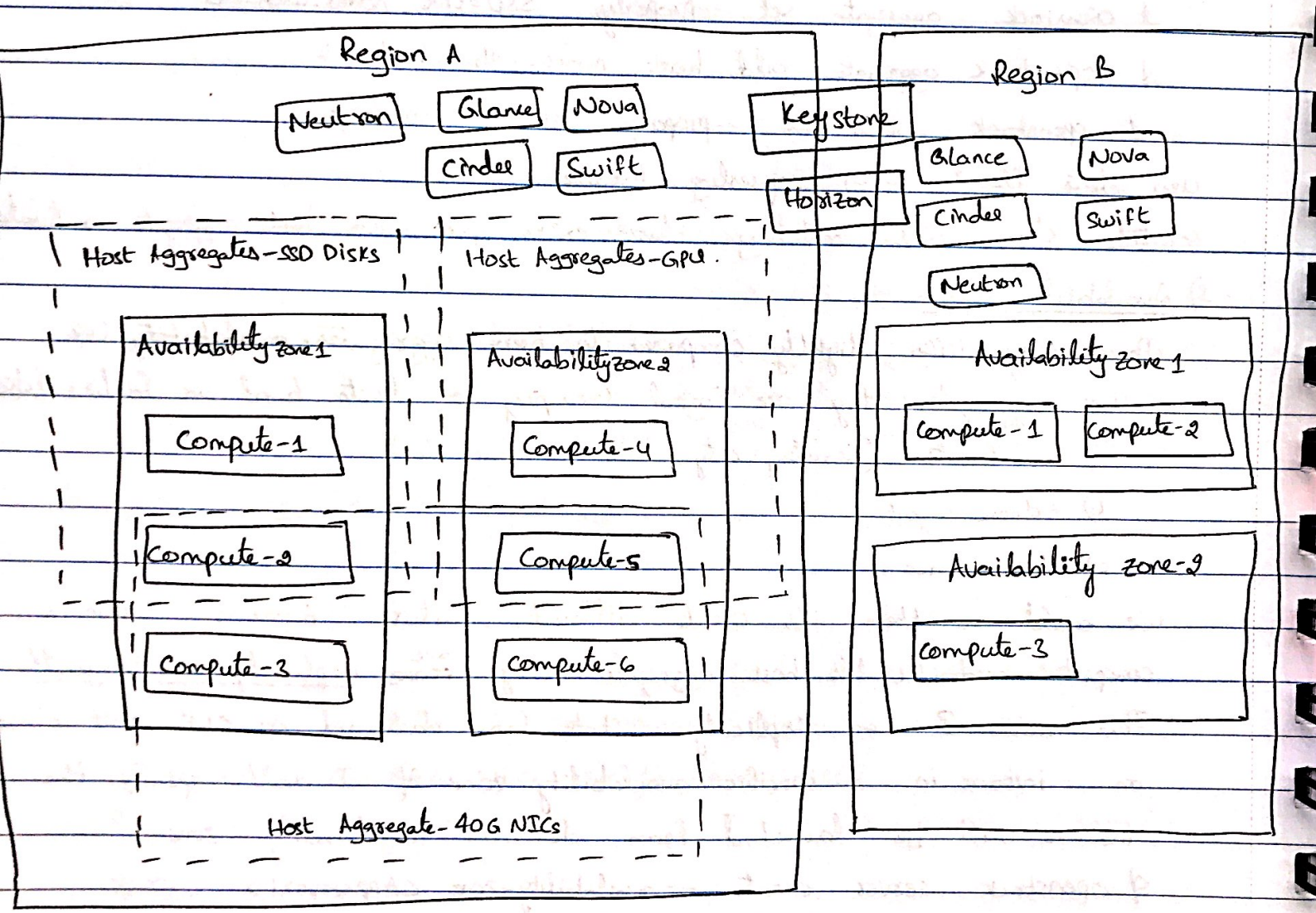
$ openstack server create --availability-zone <AppServer1>

* If we look at it clearly Availability zone is a host aggregate. It's effectively a host aggregate with special metadata on it to create an availability zone or an aggregates creates command as before.

$ Openstack aggregate create --zone <myzone> <myaggregate>

When we are dealing with availability zones, there are differences compared to host aggregates. Unlike, a host aggregate a host cannot be in multiple availability zones. It has to be one or the other.
Also, a host is part of a default availability zone even if it doesn't belong to aggregate

Segregation Example



=> CLI Novar

1) openstack compute service list (To list out nova services & their states)
2) openstack flavor create --id <ID --ram <ramsizeinMB> --disk <disk-size-inGB>
   flavor available to other projects ← --Public <name>
   " not " " " " ← --Private <name>
3) openstack flavor list.
4) openstack keypair create mykeypair >> mykeypair.key (This creates a keypair)
5) openstack image list (Lists out all the images available in Glance)
6) openstack network list (" " " Networks available)
7) We have everything we need so let's go for booting the instance.

openstack server create --image <image_name> --key-name <keypair_name>
--flavor <flavor_id> --nic net-id=<network_ID> <instance_name>

8) openstack server show <instance name or UUID> ↓

. If we have multiple compute nodes, on which compute host should that instance be launching?

The answer is Nova scheduler service determines on which host a VM should launch. There is a filtering mechanism on Nova scheduler that decides which hosts/compute nodes in our cluster are capable of launching the instance.

Among those hosts, weighing Algorithm which is again running on scheduler decides which one is best for a specific instance.

* For configuring these filters & weights Nova.conf file under etc/nova is used. Cat /etc/nova/Nova.conf |grep filter.

9) For creating snapshot we run=>
   openstack server image create --name <snapshot01> <instance01>
10) For creating an aggregate which has property SSD=true =>
    openstack aggregate create --property SSD=true

11) To get to the console of the instance

openstack console url show --nounc <instance_name>

↓

This provides a link for the console. We can paste it in our browsers and access the instance console.

12) If we want to see console logs of our instance

openstack console log show <instance_name>

13) openstack command list | grep openstack-compute -A 80

a) (compute agent) commands are used while adding or removing a new hypervisor to the cluster.

CINDER

→ Cinder is the block storage service for openstack. It is designed to present storage resources to end users that can be consumed by the openstack compute project Nova.

→ Cinder is the management of block storage devices & provides end users with a self service API to request & consume those resources without requiring any knowledge of where those storage is deployed.

→ Cinder does the life-cycle management of our persistent drives Meaning we can create, resize, migrate, delete a volume & so on.

→ Let's say we want to open up a VM & we don't want to lose info upon termination of the VM, then we need to have persistent D drive on it Cinder, it will take care of that for us if needed.

→ We can spin up a new VM later on & attach that persistent drive to that new VM & reach that critical data.

This isn't a shared storage, it's a block storage & there is one to one mapping b/w the volume & the VM.

11) To get to the console of the instance

openstack console url show --novnc <instance-name>

↓

This provides a link for the console. We can paste it in our browser and access the instance console.

12) If we want to see console logs of our instance

openstack console log show <instance-name>

13) openstack command list | grep openstack-compute -A 80

a) (compute agent) commands are used while adding or removing a new hypervisor to the cluster.

CINDER

→ Cinder is the block storage service for openstack. It is designed to present storage resources to end users that can be consumed by the openstack compute project Nova.

→ Cinder is the management of block storage devices & provides end users with a self-service API to request & consume those resources without requiring any knowledge of where their storage is deployed.

→ Cinder does the life-cycle management of our persistent drives. Meaning we can create, resize, migrate, delete a volume & so on.

→ Let's say we want to open up a VM & we don't want to lose info upon termination of the VM, then we need to have persistent D drive on it. Cinder, it will take care of that for us if needed.

→ We can spin up a new VM later on & attach that persistent drive to that new VM & reach that critical data.

This isn't a shared storage, it's a block storage & there is one to one mapping b/w the volume & the VM.

→ When we think about cinder, we can just think about USB drive. We can't plugin USB it to multiple computers at the same time. After plugging in our new drive we can format it to NTFS or EXT2 etc.. The disks can be HDDs or SSDs depending on our requirement.

→ Cinder also handles volume snapshots. Snapshots can be taken for block volumes also as well as instances

→ This also manages the volume types. Volume types are useful for differentiating b/w diff. back ends.
If we wonder what's going on behind the scenes it's just LVM (Linux Volume Management). ClusterFS as well as Ceph are also popular Software Based Solutions

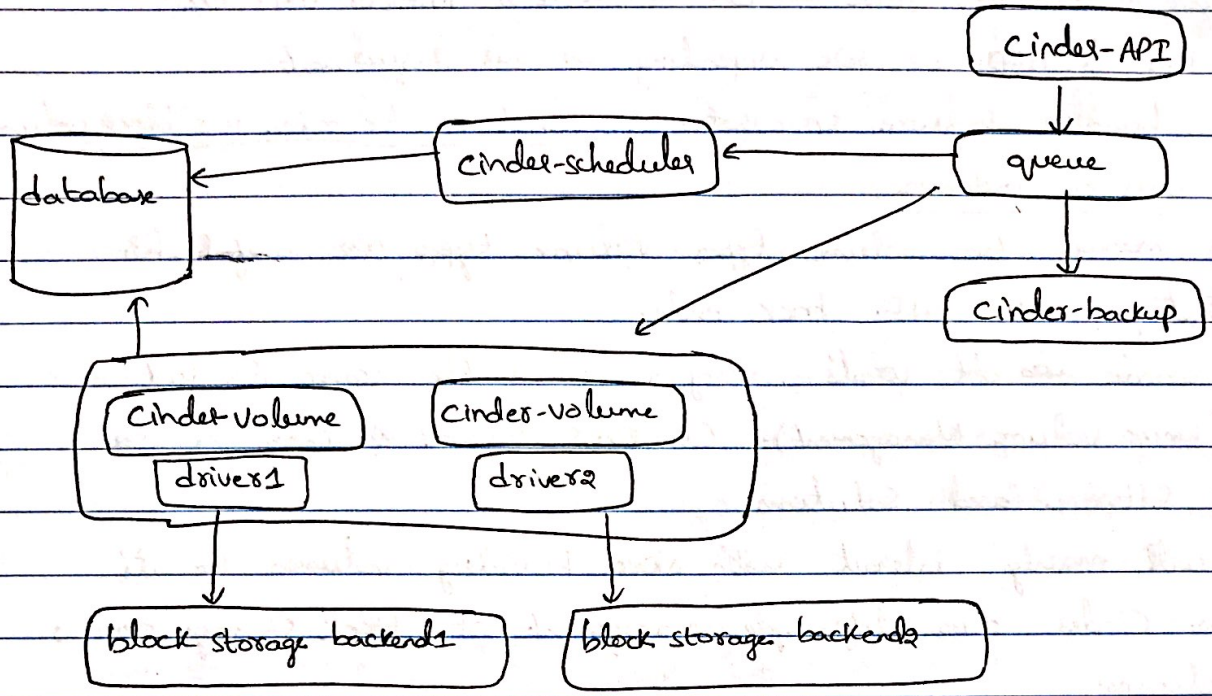
→ Cinder will mainly interact with Nova providing volumes for it's instances. Cinder does life-cycle management of block storage devices or volumes.
Volumes are block storage resources that can be attached to instances as a secondary vol storage or they can be used as the root storage to boot instances.

→ As part of it's life-cycle Management, it is possible to create, delete, extend volumes through cinder.

→ Cinder also handles snapshots. A snapshot is a read only point in time copy of a volume. Snapshots can be taken of the block volumes or of instances. We must attach a snapshot to an instance to use it as a volume.

→ A backup is a archived copy of a volume that is stored in swift objects storage.
We could backup a volume via CLI with an 'admin' user.
It is also possible to restore a volume from a previously taken backup
$ cinder backup-create "volume id"

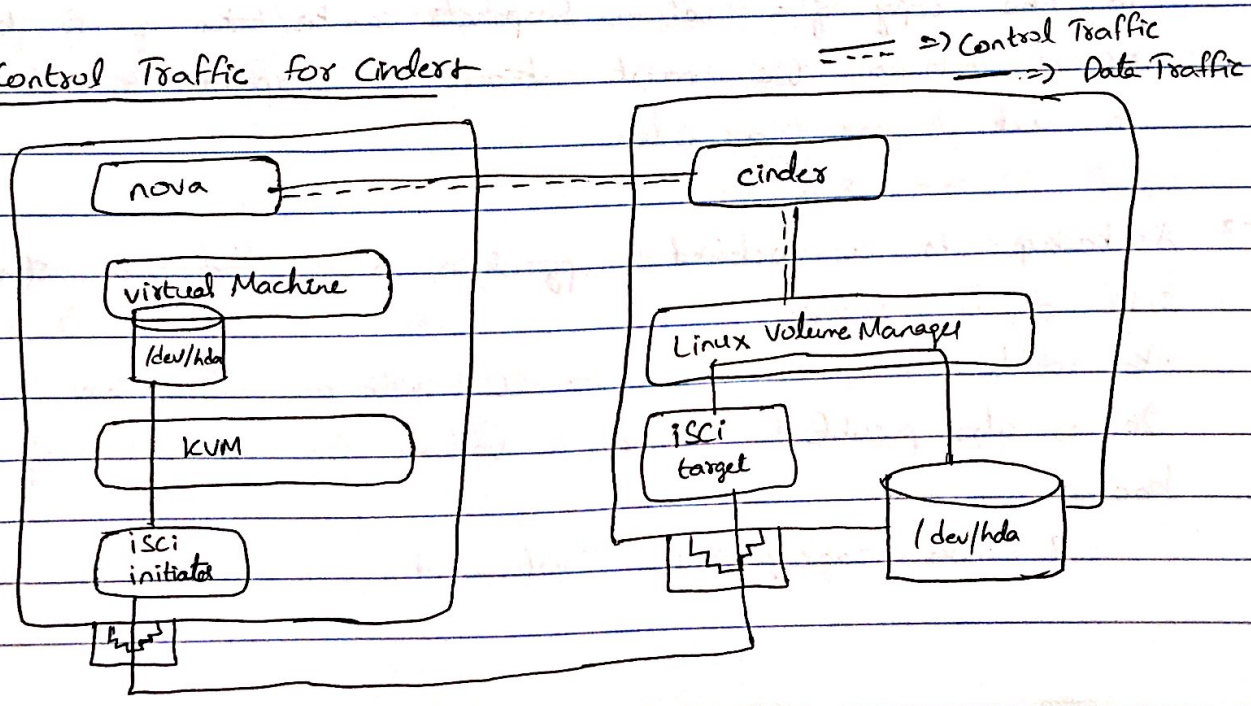
⇒ Architecture



Through API we can

1) - Volume create/delete/list/show
2) - Create from volume, image or snapshot
3) - Snapshot create/delete/list/show
4) - Volume attach/detach

Data & Control Traffic for Cinder



=> CLI Cinder

1) cinder service-list (TO see the status & list of cinder services)
2) cinder service-disable
2) openstack command list | grep openstack.volume -A 40

SWIFT

→ Swift stores objects in it.
↳ ?

An object consists of a piece of data + metadata

Swift is a pure software based solution

=> Architecture Characteristics

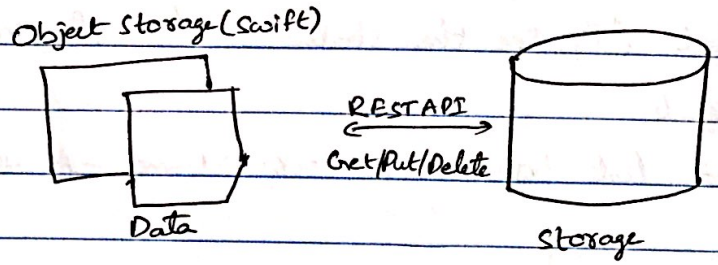
→ Swift is an eventual consistent system. eventually consistent systems are slightly different.
→ Which means when we initiate a write, multiple copies are written simultaneously
If the system is configured to write 3 copies and 2 of them complete, More than half the rights. The right is considered OK. That last failed piece will be taken care of asynchronously afterwards. We don't need to wait for all 3 pieces to complete
→ By default we have 3 replicas in Swift, we can change it.

=> * Outside applications must use RestFul APIs for communication with Swift

Eg: => https://storage.example.com/v1/AUTH_acct/cont/obj

API version — account — container — object

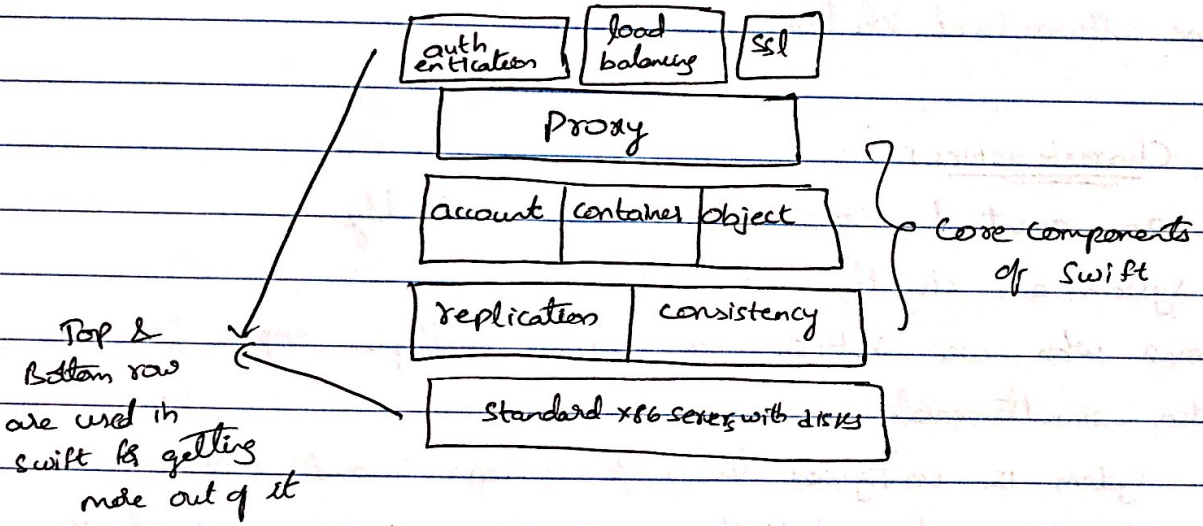
=> The Swift API:



Write an object: PUT /v1/account/container/object

Read an Object: GET /v1/account/container/object

=> System Components:



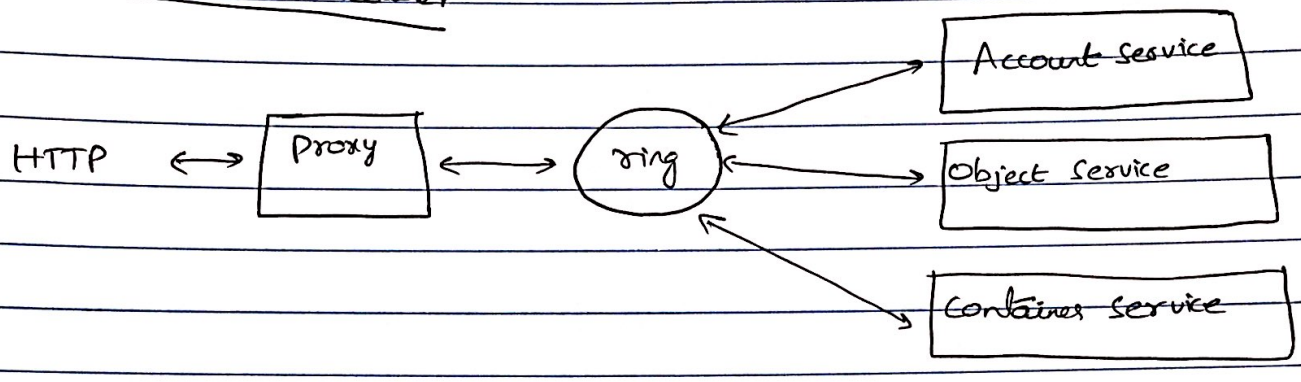
→ Proxy: Handles the requests for data read & writes

→ Account & Containers: This service provides the meta data storage service for groupings & listings.

→ Object: This service provides the interface b/w proxy servers and disk storage.

→ Replication & Consistency checks.

⇒ Swift Architecture



⇒ CLI Swift

1) openstack object store account show
(We have an account associated with our admin accounts. Every object we have will be placed under that account)
2) openstack container list (Lists all containers)
3) openstack container create <container_name> (This will create containers)
4) openstack object create -h
5) openstack object create <container_name> <file_name>
↓
for uploading this file to container
Now that file is uploaded. How to access it?
6) a) swift tempurl -h
swift post -m "Temp-URL-Key: ...."
openstack object store account show
swift tempurl get <1000> /v1/AUTH_---/container-name/file-name. <generated-k
↓
openstack This returns an output, Now,
openstack endpoint show swift (This returns publicurl)
Both combined we can use it in browser & Ta-da we can access file.
~~7) openstack command~~ b) other way is through Horizon Dashboard

7) openstack command list | grep object_store -A 20