**What is Openstack?**

<https://www.youtube.com/watch?v=Qz5gyDenqTI>

Openstack high-level overview

* You start at a web-based dashboard (**Horizon**)
* You log in (**Keystone**) and get an authentication token
* You are then presented with a JSON (Javascript object notation) catalog that presents the available resources to you
* The user is communicating with an API node
* How do the services talk? Through a queueing system called RabbitMQ.
  + A queueing system can be thought of as radio station. You subscribe to channels that you are interested in, and queueing systems allow you to “tune in” to servers or services that you are interested in. If you are interested in network events, you tune into the networking station.
    - <https://www.rabbitmq.com/>
    - What is RabbitMQ? - <https://en.wikipedia.org/wiki/RabbitMQ>
      * What is a message broker? - <https://en.wikipedia.org/wiki/Message_broker>
    - Talks about RabbitMQ Operations for Openstack – also gives some great tips on how to manage settings to improve RabbitMQ’s performance - <https://www.youtube.com/watch?v=UAUbkqMKd8s>
  + Think about a whole datacenter running on top of Openstack. Openstack is communicating with physical hardware such as disks, CPUs, RAM, switches, etc. all via messages using RabbitMQ using TCP connections. Imagine you want to spin up 5,000 virtual machines. You make an API call to Nova, Nova puts all those calls to the hypervisor into the queue (RabbitMQ), the hypervisor receives the message and spins up the virtual machines, and all of the messages get processed in order.
* The scheduler looks at the queue and the available resources and determines where resources can be distributed.

**Nova (notes from LinuxAcademy.com)**

* The nova-api accepts and responds to end user compute API calls
* nova-compute process is primarily a worker daemon that creates and terminates virtual machine instances via the hypervisors APIs (xenAPI for XenServer, **libvirt for KVM**, and the Vmware API for Vmware)
* nova-volume manages the creation, attaching and detaching of z volumes to compute instances (similar functionality to Amazon’s Elastic Block Storage)
* nova-schedule process takes a virtual machine instances request from the queue (RabbitMQ) and determines where it should run (specifically which physical server it should run on)
  + Again, RabbitMQ is the central hub for passing messages between daemons
* nova-network – manipulates the network (deprecated for Neutron)
* nova-database – SQL database that has the build-time and run-time states for the cloud infrastructure
  + Keeps track of instances in use, instances available, projects

**Swift (Notes from LinuxAcademy.com)**

* Swift is the storage layer of Openstack
* Swift is distributed storage which allows horizontal scaling\*
* Swift processes incoming requests to put data in the cluster, which results in multiple copies of the data being written to different nodes (redundancy)
* Swift processes requests to get data out of the cluster, for example to serve content to websites or to run a backup
* Swift components:
  + swift-proxy – the swift proxy server accepts incoming requests via the Openstack object API or raw HTTP. It can serve files and handles caching. The Swift proxy is the only part of Swift that communicates with the outside. All requests go to the proxy, the proxy then passes those requests on
  + Account servers manage accounts to find within the object storage service
  + Container servers manage the mapping of containers within the object storage service
  + Object servers manage actual objects (the actual files themselves on the storage nodes)
* The Swift API works over HTTP requests
  + GET – downloads objects (with metadata), lists the contents of containers, or accounts.
  + PUT – uploads objects, creates containers, or overwrites metadata headers.
  + POST – updates metadata (accounts or containers), overwrites metadata (objects), or creates containers if they don’t exist.
  + DELETE – Deletes objects or empty containers.
  + HEAD – retrieves header information, including the metadata, for the account, container, or object.

**Glance (Notes from LinuxAcademy.com)**

* Glance provides a storage and retrieval service for virtual disk images
* Openstack image service (Glance) provides discovery, registration, and delivery services for the disk and server images. Stored images can be used as a template. They can also be used to store and catalog an unlimited number of backups. The image service can store disk and server images in a variety of back-ends, including Openstack Block Storage (**Cinder**). The Image Service API provides a standard REST interface for querying information about disk images and lets clients stream the images to new servers.
* Glance administrators can create base templates from which their users can start new compute instances
* Users can choose from available images, or create their own from existing servers
* Snapshots can also be stored in the Image Service so that virtual machines can be backed up quickly
* Glance can do the following (all using an API):
  + Upload new virtual images
  + Retrieve virtual disk images
  + List available virtual disk images
  + Set permissions on virtual disk iamges
  + Querying virtual disk image properties
* Glance components:
  + glance-api – accepts image API calls for discovery, retrieval, and storage
  + glance-registry – Stores, processes, and retrieves metadata about image
  + glance-database – SQL for storing the images
  + The storage repository is usually **Swift**

**Cinder (Notes from LinuxAcademy.com)**

Cinder provides persistent block level storage devices for the use with Openstack compute instances. The block storage system manages the creation, attaching, and detaching of the block devices to servers. Block storage volumes are fully integrated into Openstack Compute (**Nova**) and the Dashboard (**Horizon**) allowing for cloud users to manage their own storage needs.

Cinder’s core features include:

* Create and delete volumes
* Create and delete snapshot of volumes
* We can attach and then deteach volumes from computer instances
* We can create new volumes from snapshots
* We can clone volumes
* Works with Glance in that we can copy images to volumes and even volumes to images
* Cinder allows us to view statistics of our volume

Volumes are raw block storage that can be attached to instances to store data persistently

Snapshots (read-only) is a point in time copy of a volume

Backups are full copies of any volume’s data and are stored in a supported backing system

Volume types allows Cinder to expose multiple backend storage providers to tenants

Cinder also incorporates quotas

* Set limits on storage (including snapshots)
* Number of snapshots stored
* Total volumes available

Cinder services

* cinder-api – daemon that accepts block storage API requests and routes them to a cinder-volume for action. It also handles authentication by routing the request to **Keystone**
* cinder-volume – acts upon requests from cinder-api and reads/writes to the cinder-database
* cinder-database – database for snapshots, volume, permissions, etc.
* cinder-scheduler – decides the optimal storage node to put the volume on. Spreads volumes across hosts equally

**Neutron (Notes from LinuxAcademy.com)**

Neutron provides true “network as a service” between interface devices managed by other Openstack services such as Nova. Neutron is a system for managing networks and IP addresses. Openstack networking ensures the network will not be the bottleneck or limiting factor in a cloud deployment and gives users real self-services, even over their network configurations.

Neutron provides a way for organizations to relieve the stress on the network in cloud environments to make it easier to deliver networking as a service in the cloud. It’s designed to provide a “plugin” mechanism that will provide an option for network operators to enable different technologies via the Quantum (soon to be changed to Neutron) API. It also lets tenants create multiple private networks and control the IP addressing on them. As a result of API extensions, organizations have additional control over security and compliance policies, QoS monitoring and troubleshooting, as well as the ability to easily deploy advanced network services, such as a firewall, intrusion detection or VPN.

* It allows users to create their own networks and then attach server interfaces to them
* Pluggable backend architecture lets user take advantage of commodity gear or vendor supported equipment
* Extensions allow additional network services, software or hardware to be integrated.

Neutron has many core network features that are constantly growing and maturing. Some of these features are routers, virtual switches and software-defined networking controllers.

Neutron concepts include:

* Ports – ports in Neutron refer to virtual switch connections. These connections are where instances and network services attach to networks. When attached to subnets, they define MAC and IP addresses of interfaces plugged into them.
* Networks – Neutron defines networks as isolated Layer 2 network segments. Operators will see networks as logical switches implmented by Linux bridging tools, Open vSwitch or some other software. Unlike physical networks, this can be defined by either operators or users in Openstack.
* Routers – provide gateways between networks.
* Private and Floating IP addresses – Private and floating IP addresses assigned to instances. Private IP addresses are visible within the instance and is usually part of a private network dedicated to a tenant. This network allows tenants instances to communicate while isolated from other tenants.
* Network Services – In addition Neutron’s low level orchestration of layer 1 through later 3 components, like IP addresses, subnets and routers, it can also manage higher level services. For example: Neutron provides load balancers as a service that utilizes the ha-proxy to distribute traffic between multiple compute instances.

Neutron services:

* quantum-server – accepts API requests and routes them to the appropriate plugin for action
* quantum-plugins – performs work of plugging in and unplugging ports
* quantum-agents – creates networks, subnets, and handles IP addressing

**Horizon (Notes from LinuxAcademy.com)**

Horizon is Openstack’s web-based dashboard. Horizon provides administrators and users a graphical interface to access, provision, and automate cloud-based resources. The design allows for third party products and services, such as billing, monitoring, and additional management tools. Service providers and other commercial vendors can customize the dashboard with their own brand.

Horizon gives the user and or administrator complete control over every aspect of Openstack – managing networks, to volumes, backups, images, Etc.

**Keystone (Notes from LinuxAcademy.com)**

Keystone provides a single point of integration for authentication for Openstack. Openstack Identity (Keystone) provides a central directory of users mapped to the Openstack services they can access. It acts as a common authetication system across the cloud operating system and can integrate with existing backend directory services like LDAP. It supports multiple forms of authentication including standard username and password credentials, token-based systems and AWS-style (i.e. Amazon Web Services) logins. Additionally, the catalog provides a query-able list of all of the services deployed in an Openstack cloud in a single registry. Users and third-party tools can programmatically determine which resources they can access.

Functions of Keystone Identity Service:

* Authenticate users and issue tokens for access to services
* Authorize users via role-based access control (RBAC)
* Provide a catalog of services (and their API endpoints) for the cloud
* Create and store policies that services can use to authorize users requests

Four primary sets of functionality in OpenStack Identity service:

* Tenants and projects
  + Keystone uses a role-based system that combines the concept of users and tenants
  + Users are the smallest unit of identity in Openstack
  + Groups organize users and roles for easier administration
  + Projects (also referred to as “tenants”) are the groups of users. This is so we can isolate customers, containing one of many users for authorization and admin purposes.
* Authentication
  + This is the primary focus of Keystone – authenticating users. Are you who you say you are?
  + It will do this by accepting an authentication request and checking it against a backed store for accuracy.
    - Example: Accepting username and password request from a user and verifying it with username and password that is stored in the backend MySQL database.
    - Note: MySQL is just one example of a backend database. There are a variety of backends to confirm this identity.
* Authorization
  + After determining that a user is who they say they are, are they now allowed to do what they’ve asked to do?
  + Authorization is handled through a rule-based engine that governs API actions. For example: Does this user have administrative access?
* Tokens
  + Once Keystone has authenticated and authorized a user’s credentials, a temporary token is issued to the user. This token can be used to request services from other OpenStack programs. The other Programs verify the token without having to process or handle the user’s credentials. The services can also use this token to even requst services from other Openstack programs on behalf of the user.

Service Catalog

* The meaning of “service catalog” in OpenStack parlance can be confusing if you come from a traditional operations background based on ITIL standards (Information Technology Infrastructure Library).
* In OpenStack, the service catalog provides a reference to the service offerings in the cloud. This could be the name, type, description, endpoint, and region for a service. OpenStack clients use this information to itelligently request services from the cloud.