This work was presented at the 2019 conference of the Information and Telecommunications Education and Research Association - ITERA 2019, Indianapolis, IN , USA, April 13-15 /2019

OpenStack Private Cloud Setup based on Virtual Machines

Ankita Kalita and Harshawardhan R. Belsare
School of Information Studies – MS in Enterprise Data Systems Program
Syracuse University

Table of Contents

1. Introduction

Many organizations are trying to move to the cloud from their legacy data centers for the scalability, flexibility and automation that can be achieved in a cloud based environment. Cloud services are more than the virtualization of computational resources. They have five essential characteristics as per the National Institute of Standards and Technology (NIST): on-demand self-service, broad network access, resource pooling, rapid elasticity, and measured service. There are three service model in cloud computing- infrastructure as a service (IAAS), software as a service (SAAS) and platform as a service (PAAS). NIST also describes four deployment models to access these services – public cloud, private cloud, community cloud and hybrid cloud. OpenStack is an open source software managed by the OpenStack foundation for creating private and public clouds. OpenStack software mainly provides Infrastructure as a Service capabilities with a shared pool of compute, storage and network resources [3].

In this paper we will present how OpenStack is used to setup a multinode private cloud. The setup was implemented and tested in a University laboratory. This paper gives a brief description of the most important services of OpenStack followed by the installation steps and then the comments on future work.

2. OpenStack components

There are different components for OpenStack for compute, network, storage, orchestration, web frontend etc. In this section several OpenStack components are described which were deployed in the environment we prepared.

2.1 Keystone - Identity service

OpenStack identity is a shared service for authorization, authentication and auditing. It also provides identity information to the end users. The project name for the identity service is called keystone. Other OpenStack services communicate with Keystone for authorization of user requests. There are mainly 3 components in the identity service: Server, Drivers and Modules.

Using a RESTful interface, a centralized server provides services for authentication and authorization. The Drivers are used for accessing identity information. The modules receive service requests, extract credentials and communicate with our centralized server for authorization [6].

2.2 Glance - Image service

The Virtual Machine OS image is a template to create new instances in the cloud. The image can only contain operating system or may also contain software and applications installed on it. Images can be created from running OS which can be used to deploy new VMs of that type.

The Glance service in OpenStack provides the capability to query OS image metadata and retrieve OS images using RESTful APIs. Glance service accepts requests from end user or OpenStack compute components through API requests.

There are 5 main components for glance:

glance-api: The role of glance-api is discovering of images, retrieving and storing images by accepting API calls.

glance-registry: Glance registry is to provide details about the metadata of the images like type, size etc. It also responsible for storing, processing the metadata details.

Database: The database is used to store the metadata of the images. In our case it is MySql.

Storage repository for image files: OpenStack supports various storage types to store image files. The images can be stored in a normal file system, RADOS block devices, Object Storage, HTTP or Amazon S3.

Metadata definition service: This service is an API to define custom metadata. This can be used by different services like – image, flavor, volumes etc. It includes a key which describes the attributes like type, description, constraints [7].

2.3 Neutron – Networking Service

The Networking service in OpenStack is known as neutron. It provides the virtual networking components in the environment and creates an abstraction between the physical network and virtual network. Other network features like firewall services, virtual private network (VPN), load balancer etc. can be enabled as well.

The virtual machines created in OpenStack can be given access to internal or external networks or both. An external network is accessible from outside the OpenStack installation. However, the internal network is accessible inside the OpenStack setup and connects directly to the VMs. Each virtual router in the environment has an interface connecting to the external network. To access VMs from the Internet, external IPs should be allocated to the VMs. To control network access to VMs, Security groups are assigned to them. These Security groups are a set of access control rules.

The network in an OpenStack environment can be configured in 2 different ways- Provider network and self-service network. In the provider network, the VMs are only attached to the provider or external network. In a self-service network configuration along with external network, VMs can have private networks and floating IPs which provide connectivity to the VMs from the external network.

The main components in Neutron are:

Neutron-server: It accepts the API requests and routes them to the appropriate network plug-in to perform specific actions.

OpenStack Networking plug-ins and agents: Layer 3 agent (L3), DHCP agent and plug-in agents are the common agents in the OpenStack environment.
Messaging queue: Keeps track of the routing information between the neutron-server and different plug-ins. It also stores the plug-ins state information [8].

In our OpenStack set up, we had to configure two virtual interfaces for the VM containing the neutron service. One interface was used to communicate with other OpenStack services and the other interface was NATed with the KVM (Hypervisor) interface. This was done to provide internet access to instances created within OpenStack. Neutron gives an internal IP address to instances then that address is NATed with the KVM interface address which enables it to get internet access.

2.4 Nova – Compute Service

Nova is a one of the crucial services of OpenStack. It interacts with the hypervisor to spin up and activate virtual machines that can be made accessible within an enterprise and supported within OpenStack. It is like an interface between OpenStack and the hypervisor. In our set up, nova services are running on two virtual machines. Hence when new VMs are created they will be inside those virtual machines (a nested VM environment – VMs inside VMs). Hence, we enabled nested virtualization on ostack01. All computing resources like RAM and vCPUs are allocated by nova from its local hypervisor.
The main components of the nova service are as follows:
Nova-consoleauth: This service provides authentication for nova consoles. This service runs on the controller node.
Nova-scheduler: Nova uses nova-scheduler to determine how to deploy instance requests. The scheduler service determines on which host the requested VM should be launched. The scheduler service depending on the resources requested for the VM, selects the physical hosts which can provide those resources else an error is generated. This service runs on the controller host.
Nova-conductor: This service provides coordination and database query support for nova. This service also runs from controller hosts.
Nova-compute: This service manages all VM related processes. This service is responsible for creating disk image, launching it via underlying virtualization driver etc. This service runs form the compute node [9].

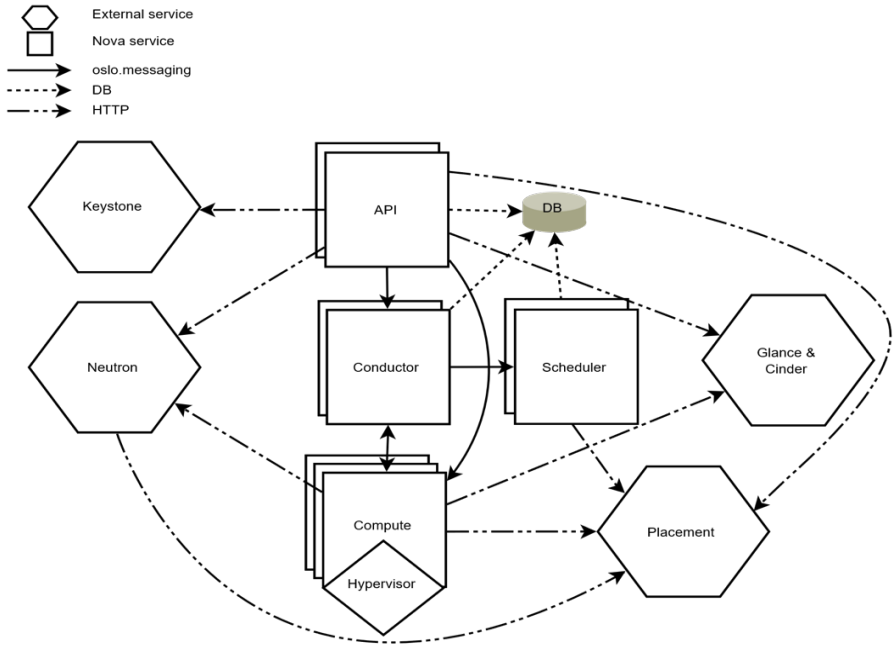


Figure 1: Primary Nova Components and It's interaction with neutron, glance, keystone
Source: https://docs.openstack.org/nova/latest/_images/architecture.svg

2.5 Cinder – Block storage Service

This service provides block storage to the OpenStack virtual machine instances. The storage is added to the VM as a separate disk. The cinder block storage service provides volumes that can be attached to each VM instances. In our deployment we have added a virtual disk to the volume group for cinder. The fdisk command set can be used to make the virtual disk as a 'PV' (physical volume). Then add this PV in the cinder volume group. Cinder also has the option of creating bootable volumes.

The main components of this service are as follows:

Cinder-scheduler: Like nova-scheduler this service selects the optimal storage host/node to create the requested volume. This service uses several filters to select the host. This service runs from the controller node.

Cinder-volume: This service runs on the storage node and manages volumes on the storage node [10]

2.6 Horizon – Dashboard

The Horizon service provides a dashboard (a web interface) for cloud administrators to manage and configure cloud resources for users. This service runs on the controller [11].

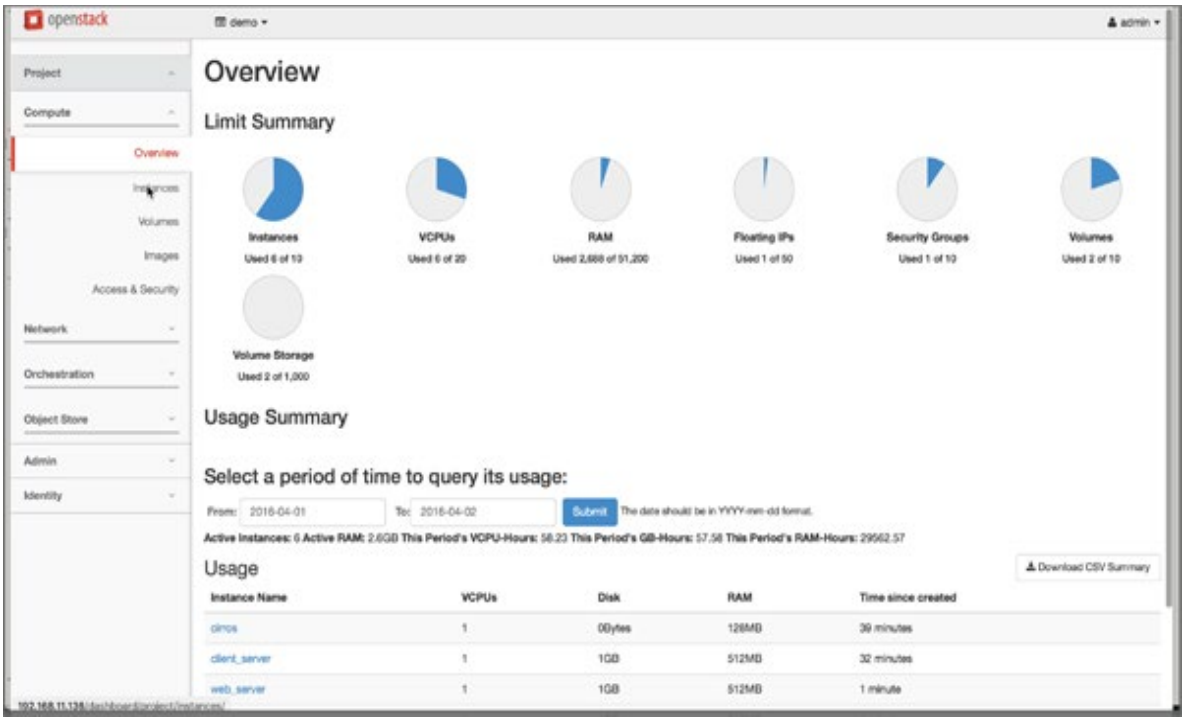
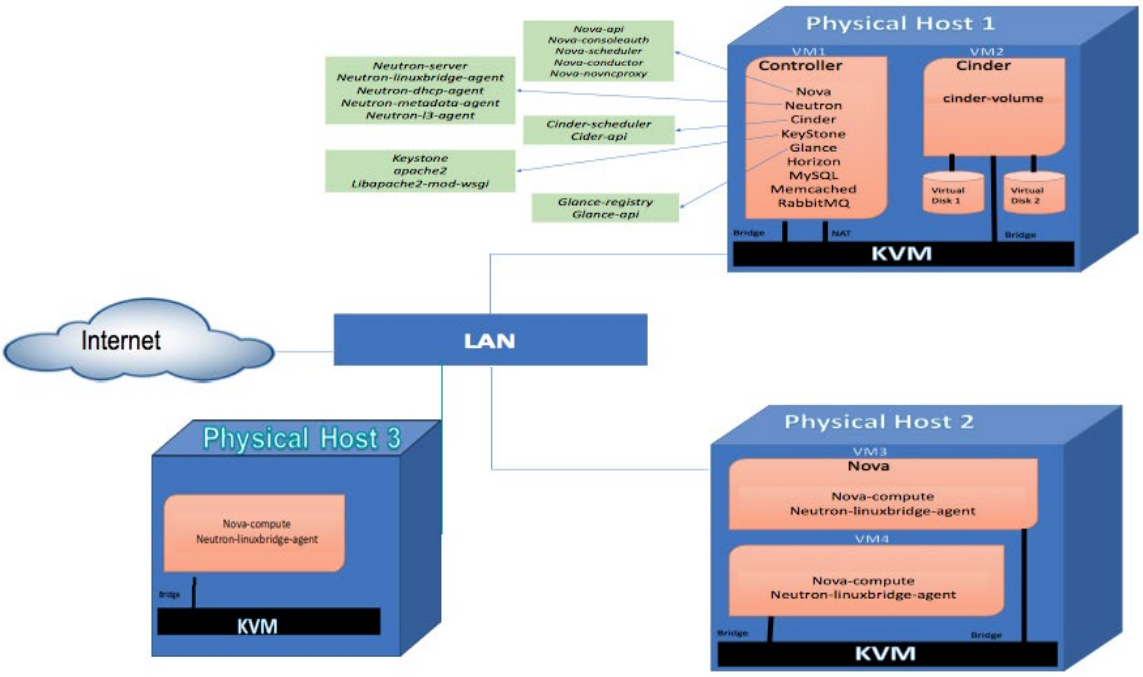


Figure 2: OpenStack Dashboard showing resources summary
Source: https://www.openstack.org/software/mitaka/

3. Installation

3.1 Private cloud architecture



The minimum requirement to install OpenStack is:

- For controller node 4GB RAM, 1 CPU and 5GB of storage
- For compute node 2GB RAM, 1 CPU and 10GB of storage

As per the architecture, different OpenStack components or set of components are to be installed on independent machines, basically one controller node, three compute nodes and 1 block storage node. This requires five machines with the hardware requirement mentioned above, either physical or virtual. The available resources we had at that moment are 3 machines with 8GB of RAM, 8CPU and 100GB storage.

As we had only 3 physical machines at that moment, we went for a nested virtualization setup, with Ubuntu server 16.04 with KVM installed on all the three physical hosts. We then deployed 2 virtual machines on two physical hosts (ostack01 & ostack02) each. One virtual machine on ostack02 contains controller and OpenStack services keystone, glance and neutron. The second virtual machine has cinder. Two virtual machines on ostack01 have nova. One physical machine ostack03 is solely used to deploy nova to utilize all its resources for the VMs that will be deployed above the OpenStack cloud environment.

The below table gives the details on the specifications of the virtual machines or physical machine along with the OpenStack services deployed on them:

physical host	VMs	Specification	OpenStack Service
ostack02 (192.168.8.4)	Controller (192.168.8.5)	4GB RAM, 2 CPU, 5GB storage	Controller, Keystone, Glance, Neutron
	Cinder (192.168.8.8)	4GB RAM, 2 CPU, 5GB storage	Cinder
ostack01 (192.168.8.31)	Nova (192.168.8.7)	4GB RAM, 2 CPU, 15GB storage	Nova
	nova1(192.168.8.28)	4GB RAM, 2 CPU, 15GB storage	Nova
ostack03 (192.168.8.32)	NA (physical host contains nova)	8GB RAM, 8 CPU, 100GB storage	Nova

3.2 Steps followed to install KVM

The KVM (kernel based virtual machine) is an open-source virtualization platform which enables an underlying Linux based machine to act as a hypervisor to deploy other VMs on top of it. The underlying CPU of the server should support virtualization. If the below command returns non-zero value, then kvm can be installed:

egrep -c '(svm|vmx)' /proc/cpuinfo

Run the below command to install KVM, virt-manager which is the graphical interface:

sudo apt-get install qemu-kvm libvirt-bin bridge-utils virt-manager

Add the users needed to use KVM.

sudo adduser <user name> libvirtd

An example for the *user name* could be: Administrator

The link referred for this installation can be found in the reference [1].

3.3 Steps to enable nested virtualization

Nested virtualization enables a system to create VMs inside VMs using the underlying hardware. As the VMs will be created inside Nova we enable nested virtualization in ostack01 and ostack03 hosts. To enable nested virtualization the following steps were followed:

1. Unloaded the kvm_probe module:
   #modprobe -r kvm_intel

2. Activated nested virtualization:
   #modprobe kvm_intel nested=1

3. Enabled nested virtualization permanently by adding below in the file /etc/modprobe.d/kvm.conf:

   options kvm_intel nested=1

4. Opened virt-manager for host 2 and configured nested virtualization on the 2 VMs nova and nova1:
   Show virtual hardware details -> CPUs -> Configuration -> Copy host CPU configuration checked box -> Apply

More information for this installation can be found in reference [2]

3.4 Detailed Installation Guide links:

We have followed the installation guides for the mitaka release to configure an OpenStack private cloud in our environment. The detail architecture was mentioned above in section 3.1 of this document.

3.4.1 Environment:

To make the environment ready to install OpenStack as described in the architecture diagram, we did several configuration changes in each of the machines (4 VMs & 1 physical) created by following the information in the following links:

1.1 Host networking:
https://docs.openstack.org/mitaka/install-guide-ubuntu/environment-networking.html

1.2 Network Time Protocol (NTP):

https://docs.openstack.org/mitaka/install-guide-ubuntu/environment-ntp.html

1.3 OpenStack packages:

https://docs.openstack.org/mitaka/install-guide-ubuntu/environment-packages.html

1.4 SQL database:

https://docs.openstack.org/mitaka/install-guide-ubuntu/environment-sql-database.html

1.5Message queue:

https://docs.openstack.org/mitaka/install-guide-ubuntu/environment-messaging.html

1.6 Memcached:

https://docs.openstack.org/mitaka/install-guide-ubuntu/environment-memcached.html

3.4.2 Identity service:

https://docs.openstack.org/mitaka/install-guide-ubuntu/keystone.html

3.4.3 image service:

https://docs.openstack.org/mitaka/install-guide-ubuntu/glance.html

3.4.4 Compute service:

https://docs.openstack.org/mitaka/install-guide-ubuntu/nova.html

- To create multiple compute nodes (nova) only *hostname* should be added in the */etc/hosts* file of the controller for the new compute node. No additional change is required in the controller. Restart the compute services in the controller node once the configuration is done in the new compute node as per the below links.
  https://docs.openstack.org/mitaka/install-guide-ubuntu/nova-compute-install.html
  https://docs.openstack.org/mitaka/install-guide-ubuntu/neutron-compute-install.html

3.4.5 Networking Service:

https://docs.openstack.org/mitaka/install-guide-ubuntu/neutron.html

- As the installation is done for self-service network hence follow:
  "Networking Option 2: Self-service networks" in the guide after installing the prerequisite.

3.4.6 Dashboard:

https://docs.openstack.org/mitaka/install-guide-ubuntu/horizon.html

3.4.7 Block Storage:

https://docs.openstack.org/mitaka/install-guide-ubuntu/cinder.html

3.6 Issues Faced and Solutions

During the installation process we came across few challenges which we like to mention and approaches we took to overcome them. Most of the issues are small and can be fixed with few additional steps. These issues may be specific to the architecture we followed, so do follow our suggested changes only if the same issues occur in your setup.

- Loopback IP should be commented in /etc/hosts, else services will start in the loopback address.
- Population of the service database sometimes failed. Eg:

If the below command fails for keystone:

su -s /bin/sh -c "keystone-manage db_sync" keystone

Solution:

run below in the mysql database:

ALTER SCHEMA keystone DEFAULT CHARACTER SET utf8;

- For Cinder, add virtual disk to the cinder node, then run the below command to make it visible:

- ✔ fdisk /dev/sda
- ✔ use t
- ✔ then w
- ✔ then pvcreate as per the documentation.
- For neutron node use additional NAT interface along with the bridge that allows the virtual machine to access internet.
- Check the mysql mariadb bind address in the file,: /etc/mysql/mariadb.conf.d/50-server.cnf
    - It should be the IP address of the controller node for the keystone database

4. Automation in OpenStack

OpenStack environments can be managed by the dashboard service called 'horizon' and by command line interface. These are effective options but can be time consuming and tedious to work with when used to manage and deploy OpenStack at large scale. For managing OpenStack at scale, we need automation to speed up some repetitive processes like configuring similar virtual machine instances, creating networks and volumes at scale, etc.

There are several ways by which we can do automation in OpenStack. One of the easiest ways is by using bash scripts. We can have OpenStack commands added in the script which when executed make API calls to different OpenStack services with given API credentials. We can also use some automation tools like Ansible to automate OpenStack deployments. Automation in OpenStack can be helpful when we want deployments at scale like configuring large number of instances at a time, configuring networks faster etc. It can also be helpful in managing OpenStack environment by automating the generation of performance reports.

OpenStack also has a service called 'heat' which is useful for automation and orchestration. In this service, the user can write a human readable template which then leverages the api structure of OpenStack to manage the deployment of VMSs and services inside the OpenStack infrastructure [4].

5. Future work – Container-based deployment

Containers package software into standardized units for development, shipment and deployment. It contains code, runtime, system tools and libraries and settings to run as a standalone executable package. They make software applications more portable and standard [5]. Where virtual machines provide an abstraction of the compute resources, containers provide an abstraction for software from the underlying infrastructure.

All the different services in OpenStack are installed independently one by one to setup the multimode private cloud environment described this document. This is time consuming and may lead to errors if any step is missed or configured wrongly. To overcome these limitations container-based installation is the solution. It will be a one time effort and once done containers can be used to spin up many cloud environments within very little setup time. There are already some frameworks for container-based deployment and our future work will be focused on how we can use our current OpenStack deployment knowledge to make container-based deployments more flexible for user specific requirements.

References:

1. "How To Install KVM And Create Virtual Machines On Ubuntu." Howtogeek.com. N. p., from https://www.howtogeek.com/117635/how-to-install-kvm-and-create-virtual- machines-on-ubuntu/
2. Fedora Quick Docs | Fedora Quick Docs | Using nested virtualization in KVM https://docs.fedoraproject.org/en-US/quick-docs/using-nested-virtualization-in-kvm/index.html
3. What is OpenStack, https://www.openstack.org/software/
4. Heat - OpenStack. Wiki.openstack.org., from *https://wiki.openstack.org/wiki/Heat*
5. What is a Container. Docker.com. from https://www.docker.com/resources/what-container
6. OpenStack Docs: Identity service overview. Docs.openstack.org. From *https://docs.openstack.org/mitaka/install-guide-ubuntu/common/get_started_identity.html*
7. OpenStack Docs: Image service overview. Docs.openstack.org. from *https://docs.openstack.org/mitaka/install-guide-ubuntu/common/get_started_image_service.html*
8. OpenStack Docs: Networking service overview. Docs.openstack.org. from *https://docs.openstack.org/mitaka/install-guide-ubuntu/common/get_started_networking.html*
9. OpenStack Docs: Compute service overview. Docs.openstack.org., from *https://docs.openstack.org/mitaka/install-guide-ubuntu/common/get_started_compute.htm*
10. OpenStack Docs: Block Storage service overview. Docs.openstack.org., from *https://docs.openstack.org/mitaka/install-guide-ubuntu/common/get_started_block_storage.html*
11. OpenStack Docs: Dashboard. Docs.openstack.org. from https://docs.openstack.org/mitaka/install-guide-ubuntu/horizon.html