**Abstract**

The on-demand availability of computer system resources, particularly data storage and computational power, without direct active supervision by the user is known as cloud computing. We can scale the resources up or down in accordance with the expansion of our firm. A sizable vendor oversees and provides all the resources, and it's typically a pay-as-you-go service. Using cloud computing to reduce unnecessary corporate investment is an effective strategy. One of the free open cloud computing systems, OpenStack gives its customers access to virtual servers and other resources. We'll use the OpenStack project in this project to deliver some of the fundamental services.

**Environment and Hardware Configuration for VMWare**

**Environment Setup**

For this project we are required to install a VM Hypervisor in the system. Here we used VMware PRO 17.0 hypervisor in the system for running the virtual machines. We also need to download the Ubuntu 20.04 LTS server iso file with SSH configuration to install it in all the virtual machines. As the hardware environment for the project, we need to create multiple VM’s in the VMware. We created VMnet1 internal Host-Only network while using the host-only bridged connection for the host pc connection. The first Virtual Machine is named “controller “ and later created another Virtual Machine named “compute” has been created. After creating the controller node, we booted into controller and started Openstack Installation Guide (Yoga) as it is the latest supported environment for Ubuntu 20.04. The SSH configuration allowed remote login with "PuTTY", which makes streamlining the OpenStack installation guide to the virtual machines practical. All installation had been done in "root" access mode as system configuration files would be updated regularly within the installation.

Figure 1 denotes the hardware settings for the “controller” virtual machine. The hardware had been configured to hold multiple resources on its original 60GB volume and enough memory to perform actions such as boot, installation, and queries. The custom (VMNet1) network adapter has been configured in VMWare “Virtual Network Editor” as shown in Figure 2. Configuring the broadcast address to 10.1.1.0 allowed for internal communication of all virtual machines using VMNet1 and the Bridged (Automatic) connection allowed the host PC to SSH into the virtual environment via PuTTY software.

**Figure 1**

*“Controller” Virtual Machine Hardware Configuration*

Graphical user interface, text

Description automatically generated with medium confidence

**Figure 2**

*“Virtual Network Editor” IP Settings*

Graphical user interface, text, application, email

Description automatically generated

Using the common VMWare steps to create a new virtual machine, the Ubuntu 20.04 LTS server image had been loaded and employed with the login of “controller” and the password of “admin”. Figure 3 denotes the IP addresses configured correctly, as the ens33 adapter contains the bridged network connection to the host (192.168.50.133) and the internal VMNet1 address as well (10.1.1.21) on the virtual ens37 interface. The addresses for Linux systems can be found using the “ip a” command line entry.

**Figure 3**

*Using “ip a” Command to Confirm Proper IP Addressing*

Text

Description automatically generated

To set the IP address statically, the Linux configuration for the common network utilities “nano /etc/network/interfaces” had been used. Figure 4 illustrates the final setup of the network, and the commented-out sections had been used during troubleshooting the network interfaces during the OpenStack installation process. As shown, ens33 is configured as “DHCP” to retrieve an IP address from the host PC’s network. Furthermore, the \*.yaml netplan file had been updated for the same configuration as the networking routes to include the gateway address for the networks. The netplan update can be done using the “nano /etc/netplan/\*.yaml” command line entry.

**Figure 4**

*“Controller” Network Interface Configuration and \*.yaml Netplan Configuration*

Graphical user interface, text

Description automatically generated

Finally, the OpenStack uses many ascii naming conventions for the virtual environments. The host names are strings tied to the IP address scheme. As noted in Figure 5, the “/etc/hosts” configuration had been modified to utilize 10.1.1.21 as the controller machine. The 10.1.1.31 IP and compute machine would be added later in the OpenStack installation process.

**Figure 5**

*Adding String Labels to the Hosts IP Addresses*

Text

Description automatically generated

Installations required building each service in a sequential steps, as one service builds the foundation to install the next. All the installations has been done in root access mode and is described in this document and followed OpenStack’s Ubuntu guides for each service.

**OpenStack IaaS Minimal Installation Setup for Yoga in Ubuntu Server 20.04 LTS**

**1. Keystone Installation :**

Keystone is an identity service provided by Openstack. The OpenStack Identity service provides a single point of integration for managing authentication, authorization, and a catalog of services, allowing systems to validate one-another and discover where other services are within the deployment (OpenStack, 2017). The Identity service is typically the first service a user interacts with.

In this part we started the installation of the keystone identity. We have followed the installation and configuration steps for the minimal deployment of keystone in Yoga. To install keystone identity service, we need to have some prerequisites installed. The following are the steps followed:

1. As part of perquisite packages we "add-apt-repository cloud-archive: yoga "
2. Downloaded Maria SQL DB as it was not included with base Ubuntu 20.04 image
3. Then created a Keystone database by following the steps given in install and configuration guide for Keystone.
   1. This step required invoking Maria SQL with the “mysql” command line entry, then “granting all privileges” to the keystone “localhost” and a default password of “KEYSTONE\_DBPASS”
4. Install keystone command with “apt install keystone” and modify its configuration files to allow the database entry and token provider information
5. Populate the service database, which can be confirmed by accessing the keystone database via Maria SQL and using the “show tables;” command
6. Setting the administrative account environmental variables, which had been added to a shell script for persistent variables once the environment had been restarted (see Figure 6)
7. Using the proper bootstrap provided in the installation guide to update the correct ports for the “controller” host as well as the admin authentication
8. Created user "admin" using the “openstack user create –domain default –password-prompt ‘admin’” command and setting the user password as “admin
9. Repeating previous step but for the “demo” user (see Figure 7)
10. Verifying users were created with the “openstack user list” command as shown in (see Figure 8)

**Figure 6**

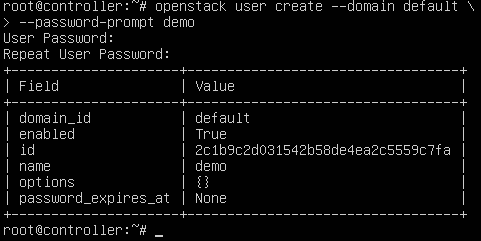
*Persistent Administrative Environment Variables*

Text

Description automatically generated

**Figure 7**

*Example: Demo User Created*



**Figure 8**

*User List Verification*

Graphical user interface, text

Description automatically generated

*Note:* Other users had been added later in the installation process. Figure 8’s screenshot had been captured post-installation of other services.

After the users “admin” and “demo” were created, roles could be assigned. Using the command “openstack role create myrole”, the users could be assigned via the “openstack role add –project myproject –user admin myrole” command. The role list, as seen in Figure 9, is verified by the “openstack role list” command.

**Figure 9**

*OpenStack Users and Their Assigned Roles*

Graphical user interface, text

Description automatically generated

Once the admin and demo users were created, tokens could be issued with the keystone service. Figure 10 denotes how the “openstack –os-aauth-url http://controller:5000/v3 --os-user-domain-name Default --os-oroject-name admin –os-username admin token issue” command can be used to issue users a token.

**Figure 10**

*Generate a Token for Admin User*

Text

Description automatically generated

**2. Glance Installation**

Glance is an image service that enables users to discover, register and retrieve the images in the Virtual Machines (OpenStack, 2017). It offers a REST API that enables us to query the virtual machine image metadata and retrieve the actual image that we are looking for. We can also store the virtual image that we obtain from image service in variety of locations.

The OpenStack Image Service includes the following components:

1. glance-api
2. Database
3. Storage repository for image files
4. Metadata Definition Service

Once we finished the installation of Keystone service, we began installing other services. For this part we started the glance image service installation on the “controller” node using the instructions.

The following steps were followed to install Glance:

1. Created database for Glance with mysql with the same commands used for keystone, but using “GLANCE\_DBPASS” for the password as prerequisite
2. Add the user “glance” as an openstack user
3. Generate an “admin” role for the glance user
4. Create the API endpoints with the ‘openstack endpoint create – RegionOne image public http://controller:9292” command, and repeat for the internal and admin endpoints.
5. Install glance with the “apt install glance” command
6. Edit the configuration files for the keystone authentication, database access, and update the quota limiting library services
7. Downloaded the "Cirros" source image and then upload to the glance image service for future accessibility and deployment, as seen in Figure 11

**Figure 11**

*OpenStack CirrOS Image Upload to Glance Image Service*

Text

Description automatically generated

1. After uploading the CirrOS image, we can validate the image is available for use by using the “openstack image list” command as noted in Figure 12. The image is now available for future virtual machine creation.

**Figure 12**

*Confirmation CirrOS Image is Successfully Uploaded*

A screenshot of a computer

Description automatically generated with medium confidence

**3. Placement Service Installation that is needed to work with Nova Installation**

For installing the Nova Service, we need install the Placement services. The placement service provides a module level application attribute that most of the WSGI servers expect to find and making it possible to run many different API configurations.

The following are the steps that were used to install the Placement service:

1. Created database for Placement with Maria mysql
2. Created a Placement user for the "default" domain
3. Created public, internal, and admin endpoints with the controller:8778 port
4. Installed and configured keystone authentication services and database information in the Placement configuration scripts. Default password had been used as “PLACEMENT\_DBPASS”
5. Before proceeding with the Nova Service Installation, we Placement working with the "placement-status upgrade check" command, as noted in Figure 13

**Figure 13**

*Placement Status Verification*

Text

Description automatically generated

**4. Nova Installation**

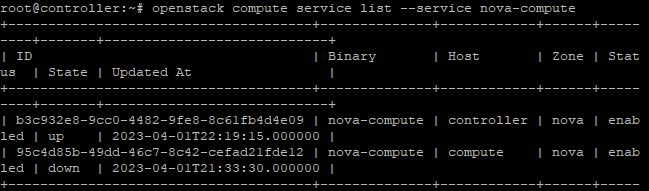
Compute (Nova) is a cloud computing fabric controller which is the main part of IaaS System. It is designed to manage and automate the pools of computer resources and can work with widely available virtualization technologies as well as bare metal and High Performance Computing (HPC) Configurations.

The following are the steps followed to install the Nova Service, specifically for the “controller” machine.

1. Install nova-compute with the “apt install nova-compute” command
2. Install RabbitMQ services for the transport layer
3. Edit the nova configuration files, updating the following:
   1. keystone authentication services
   2. IP addressing to note the host ip (10.1.1.21)
   3. Adding the vnc section, allowing remote console access for the controller:6080 port
   4. Adding the glance service configuration for the controller:9292 port
4. Verifying there are compute hosts in the database, as noted in Figure 14
5. Repeating steps 1-4 for a “compute node”
   1. Using VMWare’s “clone” tool assisted with creating the compute-node
   2. A clone of the controller had been implemented, its host-name updated to “compute”, and its IP addressing updated to 10.1.1.13”
6. Figure 14 also denotes the compute service is running while both virtual machines are in a powered-up state

**Figure 14**

*Confirmation Both Controller and Compute Hosts Were in the Database*



Once both controller and compute were created, a “virtual” virtual machine could be added with the CirrOS image available in the glance service. The process to add and login to the virtual machine are available in post-Horizon Dashboard installation guide.

Confirmed the placement API is working and prerequisites were in place with "nova-status upgrade check" command before continuing to Neutron install. The status of the Nova installation can be noted in Figure 15.

**Figure 15**

*Nova-status upgrade check output*

Text

Description automatically generated

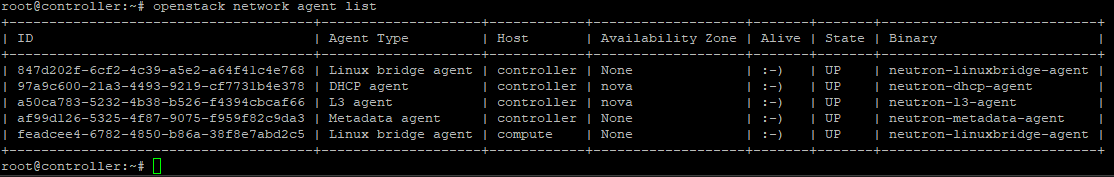
**5. Neutron Installation**

Neutron is an open stack project that is used to provide the “network connectivity as a service” between the interface devices managed by other OpenStack services. The following sequence had been used to install the Neutron networking service:

1. Created a database in Maria SQL for Neutron, with the default password “NEUTRON\_DBPASS”
2. Created a Neutron user, and granted admin role status
3. Configured the public, internal, and admin endpoints with the controller:9696 port
4. We started the installation of Neutron on “Controller” node
5. Utilized the Networking Option 2:Self-Service networks guide so to utilize access for all users to manage the provider network and floating IPs can be utilized
   1. This updated a series of configuration files for Neutron, including the ml2 plug-in, the linux bridge service, and the DHCP agent for virtual networks
   2. The previous step had been repeated for the compute-node VM
6. Edited the metadata\_agent.ini file to have “controller” as the nova host and the shared secret password as “admin” for the metadata proxy
7. Updated the Neutron section of the Nova configuration file as instructed, and used the metadata proxy password
8. Restarted all services
9. Figure 16 shows the validation of the network services and their state using the “openstack network agent list” command

**Figure 16**

*Network Agent List Post-Installation of Neutron Services*



**6. Horizon Installation**

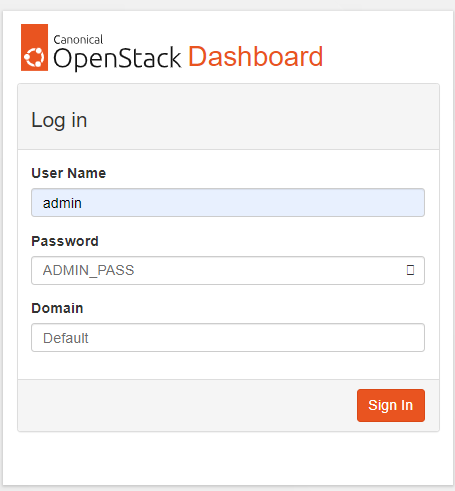
Horizon is a dashboard that provides administrators and users with a graphical interface to access, provision and automate cloud based resources. The dashboard is one of the several ways that users use to interact with the openstack services.

The following are the steps used to install Horizon Dashboard service:

1. Install openstack-dashboard for the controller node
2. Configure the local\_settings python script to use services on the controller node, update the Memcached session storage service for the controller:11211 port, and update identity services.
   1. The time zone can be set for Horizon, but was kept as the default UTC time
3. Restart apache2 services
4. Login to the OpenStack Horizon Dashboard on the host PC, using the address 10.1.1.12/horizon using the Chrome browser. The login credentials and the resulting dashboard are denoted in Figure 17and 18, respectively.
   1. This used the environmental variables setup in the keystone installation guide

**Figure 17**

*OpenStack Horizon Dashboard Login at 10.1.1.12/horizon*



**Figure 18**

*Snapshot of Browser-Based Horizon Dashboard Utility*

Graphical user interface, application

Description automatically generated

After the Horizon installation had been completed, a VM could easily be created using the utilities provided and the CirrOS image downloaded during the Glance services install. This is a “light-weight” OS utilized during the OpenStack installation. The steps below denote how the instance had been created using the Horizon Dashboard.

1. Create an “instance flavor” to specify virtual hardware profiles. Common naming conventions for the instance flavors and setup came from the redhat installation of OpenStack guide
   1. A flavor instance of “m1.small” had been used with command “openstack flavor create –ram 2048 –disk 20 –vcpus 1 –public m1.small
      1. This created a flavor with 20GB of storage, 2GB of RAM, and one virtual CPU with the name “m1.small”
2. A public and private network had been created for the “cirros” virtual machine as well with the setup in Figure 19.

**Figure 19**

*Public and Private Network Setup in Horizon Dashboard*

Graphical user interface, application

Description automatically generated

1. Next, a router with an external gateway for the public network and a port for the private network would be added as displayed in Figure 20
   1. This allows the private IP to access the public network and vice-versa

**Figure 20**

*Router Configuration with Internal Interface and External Gateway*

Graphical user interface, application

Description automatically generated

1. Now, an instance could be created. In the Project/Instance menu, the cirros image could be assigned to a virtual machine.
   1. The DHCP service on the private network assigned the new instance an IP of “192.168.1.208”
   2. A floating IP was generated, available due to the Option 2:Self-Service configuration in the Neutron install, of “10.1.1.139” for the public network interface
   3. Figure 21 confirms the installation of the “cirros” machine to be configured correctly from the Horizon Dashboard

**Figure 21**

*Cirros Instance Successfully Created*

Graphical user interface, application, Word

Description automatically generated

1. Once the instance had been created, the controller virtual machine could SSH into the machine.
   1. The default password is “gocubsgo” for the image downloaded with Glance and is the successful instance SSH is noted in Figure 22

**Figure 22**

*Successful cirros Instance Login*

Text

Description automatically generated

**7. Cinder Installation**

Cinder is a block storage that provides persistent block‑level storage devices for use with OpenStack compute instances. Block storage volumes are fully integrated into OpenStack Compute and the dashboard allowing for cloud users to manage their own storage needs.

The following steps are followed to install Cinder:

1. Created database with all privileges in Maria SQL with the default password “CINDER\_DBPASS”
2. Added the user cinder to the default domain
3. Create the cincerv3 services as well as the public, internal, and admin endpoints with the controller:8776 port
4. Configure the keystone authentication, rabbit transport layer protocols, and the IP of “10.1.1.21”
5. Update Nova configuration file’s “cinder” section
6. Install “lvm2 thin-provisioning tools” and the “cinder-volume tgt” tool
7. Install the cinder-backup service
8. Verify the cinder volume service list is correct with the “openstack volume service list” command. Figure 23 denotes the output of the service list

**Figure 23**

*Cinder Service List*

Text

Description automatically generated

Unfortunately, the volume service list did not populate as expected. After attempting steps 1-8 three sperate times, our team is unsure of the issue with the Keystone service and Cinder’s configuration. We can verify the cinder-volume services are created, as denoted in Figure 24, and a volume is most likely configured correctly using the “fdisk -l” command as noted in Figure 25. We will attempt to troubleshoot in the future with the error log information and online support.

**Figure 24**

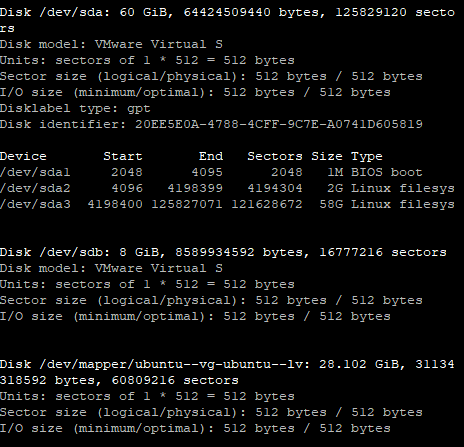
*Cinder Volume Service Available*

A screenshot of a computer

Description automatically generated with medium confidence

**Figure 25**

*Volume Is Available in Virtual Environment as Disk /dev/sdb*



**8. Swift Installation**

Swift is an OpenStack object store service which offers cloud storage software that assists in retrieving lots of data with a simple API. After installing swift, the service allows storing unstructured data that can grow without bound. For Swift’s installation, we will use the install and configuration for Ubuntu (and Debian) instructions via the OpenStack website.

The following steps were used to install Swift:

1. Create an extra storage volume on the controller VM with the VMWare Settings, which are denoted in Figure 26

**Figure 26**

*8GB SCSI Volume Added to Controller VM*

Graphical user interface, application

Description automatically generated

*Note*: Options to add volume are located at Virtual Machine Settings/Add…/Hard Disk

1. Installed utility packages, xfsprogs and rsync
2. Formatted the new hard disk with “mkfs.xfs /dev/sdb” command
3. Mount a point directory with “mkdir -p /srv/node/sdb”
4. Edit the /etc/fstab file with “/srv/node/sdb xfs noatime 0 2” for UUID
5. Mount the new sbd disk with “mount /srv/node/sdb”
6. Update the rsyncd configuration file for IP addressing and network management on the new storage node
7. Enable and restart rsync services
8. Install swift-account, swift-container, and swift-object
9. Download the account-server, container-server, and object-server configuration files, which were saved directly to the new /etc/swift directory post-swift install
10. Update the account-server, container-server, and object-server with the IP address management network, metering, and correct modules via the OpenStack Swift installation guide for Ubuntu
11. Take ownership of the mount point with “chown -R swift:swift /srv/node”
12. Create a recon directory and take ownership of the directory as well with the commands in the guide
13. Build account rings to maintain a list of containers

Building the account rings is the next step after configuring each of the configuration files. Each ring for account.builder, container.builder, and object.builder had been created directly in the /etc/swift directory for the /dev/svb storage node. Ports used for the account.builder, container.builder, and object.builder were 6200, 6201, and 6202, respectively. Once each ring had been created, rebalancing the partitions created ring configuration \*.gz files that are denoted in Figure 27. The 1-14 step process of installing Swift (and the configuration rings), except for admin features, had also been configured on the Compute node with a 10GB volume.

**Figure 27**

*Account Ring Configuration Files Created*

Text

Description automatically generated

After the account ring configuration files had been created, the final installation steps could be done for Swift. The “swift.conf” file had to be downloaded to the /etc/swift directory and updated. The Hash prefix and suffix were both updated to “admin” for their values. The storage-policy section had also been updated. Then, after ensuring proper ownership of the configuration directory, the Memcached and swift-proxy services were restarted.

Finally, when the swift configuration file had been updated, we could start the Object Storage services with the “swift-init all start”. Unfortunately, like Cinder, we had additional issues with Keystone. The keystonemiddleware python environment did not appear to connect the authention variables correctly, thus not completing the full Swift installation. This prevented containers in the next steps of the installation procedure and we were not able to complete this part of the OpenStack installation. Furthermore, troubleshooting tips do not appear to be available in the online help or through discussion forums. A request for help online could be useful but will most likely extend completion of this section beyond its due date. The keystonemiddleware exception and log are available in Figure 28.

**Figure 28**

*Keystonemiddleware error*

Text

Description automatically generated

**Extra Points Section**

The following documentation is not part of the traditional OpenStack install. These are extra point steps solely done for the assignment. The following information includes installing the “Heat” service with a public network attached.

1. **Heat Installation with Public Network**

Heat orchestrates the infrastructure resources for a cloud application based on templates in the form of text files that can be treated like code. For the installation, we followed the installation instructions for Heat (Ubuntu) in the OpenStack guide.

The following steps are followed to install Heat:

1. Created the heat database with all privileges in Maria SQL with the default password “HEAT\_DBPASS”
2. Added the user “heat” to the default domain, and granted the admin role
3. Created the services for orchestration and cloudformation with the “openstack service create --name heat --description "Orchestration" orchestration” command and the “openstack service create --name heat-cfn --description "Orchestration" cloudformation” command, respectively.
4. Created the public, internal, and admin endpoints for heat on controller:8004 port for both orchestration and cloudformation
5. Created a new domain for “Stack projects and users”, and added the user ‘heat’
6. Created the heat\_domain\_admin user for the ‘heat’ domain, and granted admin privileges
7. Create the heat\_stack\_owner role, and assign the role to the demo project to enable stack management
8. Create the heat\_stack\_user role
9. Installed the configuration components for the heat-api and its dependencies, as well as updated the configuration files as necessary
10. Populated the orchestration database and restarted the orchestration services
11. Verified the initial configuration of Heat with “openstack orchestration service list”. The output is shown in Figure 29

**Figure 29**

*OpenStack Verification Step*

Graphical user interface

Description automatically generated with low confidence

After the Heat services were correctly configured, a template file could be used to launch new instances. Per the OpenStack instructions, our team created the “demo-template.yml” file which is denoted in Figure 30. The cirros imaged and the m1.small flavor had been used for the instance creation.

**Figure 30**

*Demo-template.yml File*

Text

Description automatically generated

Once the template file was created, the network environment variable for the NetId could be configured. Switching to the demo-openrc credentials, we were able to re-visit our network scheme with “openstack network list command” which output the network information available in Figure 31. Due to our naming convention of the **public network**, we needed to create the NET\_ID environment variable with the following command which varied slightly from the OpenStack instructions:

export NET\_ID=$(openstack network list | awk '/ public-ipv4 / { print $2 }')

**Figure 31**

*OpenStack Network List Output*

Timeline

Description automatically generated

Afterwards, using we could create the instance “heat\_is\_ece530” with the following command: openstack stack create -t demo-template.yml --parameter "NetID=$NET\_ID" heat\_working\_ece530. We were then able to verify with the stack list that the instance had launched its creation successfully as denoted in Figure 32.

**Figure 32**

*OpenStack Instance Launch Successful from Template File*

A computer screen capture

Description automatically generated with low confidence