Introduction

OpenStack is a suite of services that can be used together to implement a cloud of virtual machines. What does this mean? At its heart, OpenStack provides services allow the end user to create and manage virtual networks and virtual machines and configure them in many different ways. There can even be multiple independent virtual clouds in the same deployment. There is no single “OpenStack” deployment. Each user of the system must consider the hardware that is available along with the use case to determine which services are required. The full scope of OpenStack is far beyond what can be covered in a single lab. This lab will cover most of the basics needed to set up a nearly minimal deployment.

Five key services are needed to get a minimal deployment working. At the core of everything is the identity service, keystone. This service provides authentication and authorization for all parts of the stack. The image service, called glance, manages the images that are available to launch instances of virtual machines. The placement service, called placement, tracks resource usage among the components of the system. While it is needed to set up a minimal deployment, you will not discuss it further. The compute service, called nova, manages compute instances. These are the virtual machines in the cloud. The networking service, called neutron, manages the virtual networks of the deployment and the transport of the data over the actual physical networks connected to the hardware. A final service, called horizon, is not strictly needed for a minimal deployment. It provides a web user interface for managing the virtual clouds.

There are many other services that can be included in a deployment. They typically cover more advanced management and configuration features. One service that would probably be high on the list for inclusion is called cinder, a block storage service. This service enables the management of data storage in the virtual clouds among many physical storage systems.

For this lab, consider the following situation. Your company has been using a cloud provider for IaaS. However, a new project has higher data security requirements, and there has been some concern about using an external provider. Your team has been tasked with investigating the possibility of creating your own private cloud service on local hardware. The goal is to eventually end up with a system that can be used in a way similar to the current external solution but that can be hosted locally.

The current scale of the project is not large, but it may grow in the future. The large range of capabilities of the OpenStack system seem attractive for allowing the system to grow as needed. However, your current team is not familiar with managing an OpenStack deployment. You are going to start working through configuring a nearly minimal installation to get a better understanding of the system.

There are many ways to approach learning how to deploy and manage an OpenStack system. One popular method for testing uses DevStack. This provides a script that will install the latest OpenStack version onto a single machine with some minimal initial configuration. While this is good for quick testing, it hides many aspects of the deployment in its inner workings. Your team has decided to instead work from some of the guides for a minimal OpenStack installation and deployment.

Lab Overview

This lab has **three** parts, which should be completed in the order specified.

1. In the first part of the lab, you will configure and start the identity service for OpenStack.

2. In the second part of the lab, you will configure the compute service on both a controller node and a compute node.

3. In the third part of the lab, you will configure the networking service on the controller node and start the dashboard service.

Finally, you will explore the virtual environment on your own. You will answer questions and complete challenges that allow you to use the skills you learned in the lab to conduct independent, unguided work – similar to what you will encounter in a real-world situation.

Learning Objectives

Upon completing this lab, you will be able to:

1. Configure and start the keystone services on the controller node.

2. Configure and start the OpenStack compute services on a controller node.

3. Configure and start the OpenStack compute services on a compute node.

4. Configure and start the OpenStack network services on a controller node.

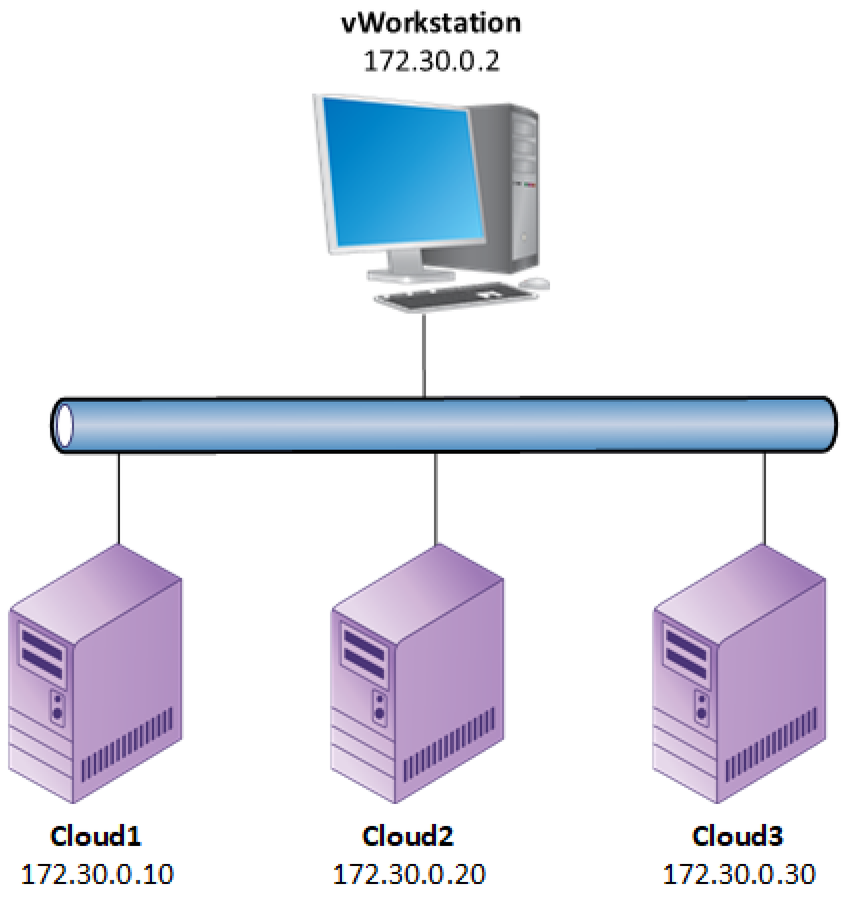
5. Configure and start the OpenStack dashboard service.

6. Describe the interoperation of several key OpenStack cloud services.

Topology

This lab contains the following virtual machines. Please refer to the network topology diagram below.

- vWorkstation
- pfSense
- cloud1
- cloud2
- cloud3



Tools and Software

The following software and/or utilities are required to complete this lab. Students are encouraged to explore the Internet to learn more about the products and tools used in this lab.

- SSH
- Vim
- OpenStack

Deliverables

Upon completion of this lab, you are required to provide the following deliverables to your instructor:

Hands-On Demonstration

1. Lab Report file, including screen captures of the following;

- Information about the test project, test user, and test role
- The generated token information
- Cell0 and cell1 in the command output
- Cloud2 as an active compute node for the stack
- The horizon login page

2. Any additional information as directed by the lab:

- None

Challenge and Analysis

1. Lab Report file, including screen captures of the following:

- Your new flavor in the flavor list output
- Both cloud2 and cloud3 in the list of available compute nodes

2. Any additional information as directed by the lab:

- None

Hands-On Demonstration

Note: In this section of the lab, you will follow a step-by-step walk through of the objectives for this lab to produce the expected deliverables.

1. **Review** the **Tutorial**.

   Frequently performed tasks, such as making screen captures and downloading your Lab Report, are explained in the Cloud Lab Tutorial. The Cloud Lab Tutorial is available from the User menu in the upper-right corner of the Student Dashboard. You should review these tasks before starting the lab.

2. **Proceed** with **Part 1**.

Part 1: Set Up Basic Services on the Controller Node

Note: For the initial testing, some other members of your team have already set up a few machines to be used for the minimal deployment. You will be working from the vWorkstation desktop and connecting to the “cloud” machines using SSH. Each of the cloud machines has Ubuntu 20.04 server edition installed. These machines have no GUI. There are packages for the major components for OpenStack for this Linux distribution. For the installed version, the packages correspond to an OpenStack version called Ussuri.

Any OpenStack deployment requires at least one controller node. For a small deployment, one should be sufficient. The controller node runs key services that are required for the whole stack to operate. Three services are provided by packages from outside the OpenStack system:

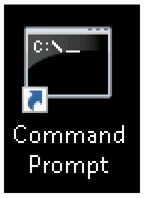
- An SQL database, provided by MariaDB
- A message broker, provided by RabbitMQ
- A caching service, provided by memcached

These have already been installed on cloud1. In addition, the OpenStack packages for the keystone, glance, placement, neutron, nova, and horizon services have been installed. The services have been disabled and will need to be reenabled once the configuration is ready.

The controller node does not actually host any virtual machines in the deployment. Instead, it hosts the services that are used to manage the system. Compute nodes can be added as needed to expand the capacity of the infrastructure based on the demand. The controller node should also have two network interfaces (NICs). Some of the reasons for this will be discussed later in the lab.

To begin, you decide to first verify that cloud1 is suitable for use as a controller node in a nearly minimal OpenStack deployment.

1. On the vWorkstation desktop, **double-click** the **Command Prompt icon** to open a command prompt window.



Command prompt icon

Note: Notice that the prompt has been set up so that the working directory is the desktop. Use this shortcut whenever you need to open a new command prompt in this lab.

2. At the command prompt, **type** `ssh cloud1` and **press Enter** to connect to the cloud1 machine using SSH.

Note: You should see a shell prompt for user@cloud1. Windows has OpenSSH installed by default. The team has set up the machines to use key pairs for authentication, so no passwords are needed for connecting. They also created a configuration file to ease the process.

3. On the vWorkstation desktop, **double-click** the **ssh config shortcut** to open the SSH configuration file in Notepad++.

SSH configuration shortcut

Note: You can see three sections starting with Host lines. Each section sets a few variables for the SSH connection. When you type *ssh cloud1*, a section starting with Host cloud1 is searched for. If one is found, the settings from that section are used. Here, the settings give the username to log in with, the IP address of the cloud1 machine, and the private key file to use for the connection. There are similar sections for the other two cloud machines.

To enable the password-less login, the public key corresponding to the private key in C:\Users\Administrator\.ssh\vm was copied to each of cloud1, cloud2, and cloud3 and placed in the ~/.ssh/authorized_keys file. You will still need to give a password to use sudo. This configuration file will also work with the scp command, so you can do things like "scp /path/on/vWorkstation alias:/path/on/remote" to copy files without being prompted for a password.

Now let's investigate the network configuration for the controller node.

4. **Close** the **Notepad++ window**.

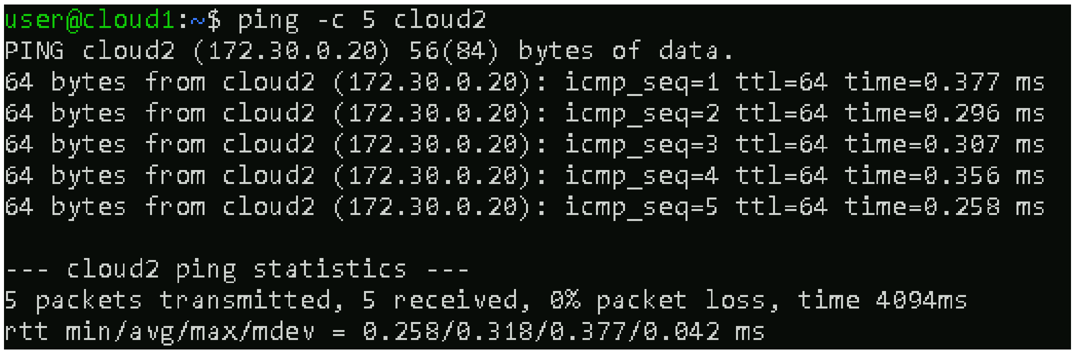
5. In the command prompt window, **type** `ip a` and **press Enter** to list all the network interfaces on cloud1.

Note: You should see three network interfaces. The lo interface is the loopback interface. The ens192 interface has the IP address 172.30.0.10. This is the interface you used to connect to the machine via SSH. For the OpenStack deployment, this will be the interface to the management network. There is a second ethernet interface, ens224, that has no IP address assigned. This will become the interface for the provider network for the OpenStack deployment. You will discuss what these two networks are later in the lab.

6. In the command prompt window, **type** `cat /etc/hosts` and **press Enter** to view the hosts file for cloud1.

Note: For simple network configurations, host names can be assigned by adding entries to the /etc/hosts file. Here you can see that entries have been added for the three cloud machines.

7. In the command prompt window, **type** `ping -c 5 cloud2` and **press Enter** to attempt to ping the cloud2 machine by name.



```
user@cloud1:~$ ping -c 5 cloud2
PING cloud2 (172.30.0.20) 56(84) bytes of data.
64 bytes from cloud2 (172.30.0.20): icmp_seq=1 ttl=64 time=0.377 ms
64 bytes from cloud2 (172.30.0.20): icmp_seq=2 ttl=64 time=0.296 ms
64 bytes from cloud2 (172.30.0.20): icmp_seq=3 ttl=64 time=0.307 ms
64 bytes from cloud2 (172.30.0.20): icmp_seq=4 ttl=64 time=0.356 ms
64 bytes from cloud2 (172.30.0.20): icmp_seq=5 ttl=64 time=0.258 ms

--- cloud2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4094ms
rtt min/avg/max/mdev = 0.258/0.318/0.377/0.042 ms
```

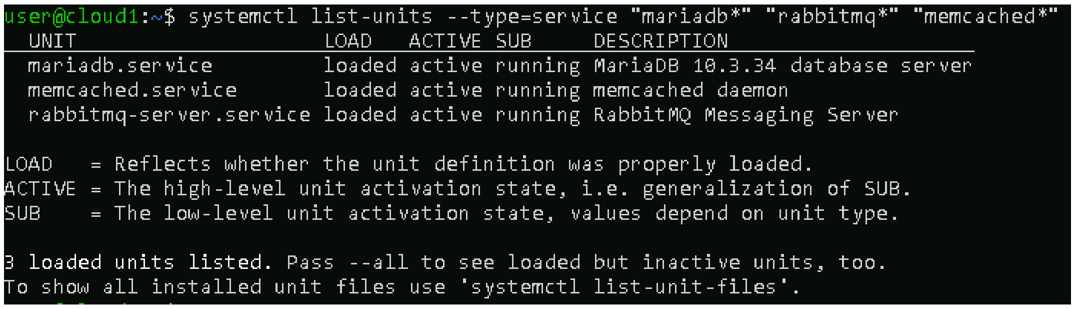
Ping command

8. In the command prompt window, **type** `ping -c 5 cloud3` and **press Enter** to attempt to ping the cloud3 machine by name.

Note: In both cases you should see five pings sent and received (the -c 5 says to try five times). Therefore, the team has set up the network to enable simpler commands to be used for connecting to the machines and for testing connectivity among them. The cloud1 machine satisfies the host networking requirements for the controller node.

Before starting to configure the controller node, you also want to verify that the required services are running. Recall that there should be a database, message broker, and cache service running. These should all be running on the controller node. Based on the notes from your teammates, these should already be installed; you just want to verify that fact. You can check the status of services using the systemctl command.

9. In the command prompt window, **type** `systemctl list-units --type=service "mariadb*" "rabbitmq*" "memcached*"`.



```
user@cloud1:~$ systemctl list-units --type=service "mariadb*" "rabbitmq*" "memcached*"
  UNIT                        LOAD   ACTIVE SUB     DESCRIPTION
  mariadb.service             loaded active running MariaDB 10.3.34 database server
  memcached.service           loaded active running memcached daemon
  rabbitmq-server.service     loaded active running RabbitMQ Messaging Server

LOAD   = Reflects whether the unit definition was properly loaded.
ACTIVE = The high-level unit activation state, i.e. generalization of SUB.
SUB    = The low-level unit activation state, values depend on unit type.

3 loaded units listed. Pass --all to see loaded but inactive units, too.
To show all installed unit files use 'systemctl list-unit-files'.
```

Command to list services

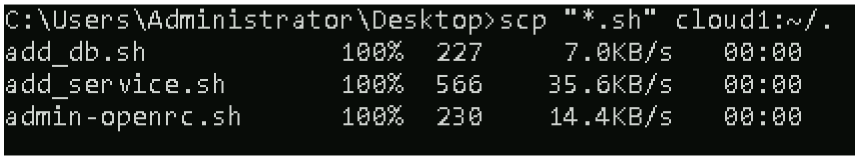
Note: The three quoted strings are patterns to use to search the service names. This command should show you a list of three services that are running.

Another member of your team has reviewed the installation guidelines and noticed that there are some very repetitive pieces among the various steps. This team member created some helper scripts to reduce the amount of work needed to do the initial test deployment. You will introduce and use these along the way. They are on the desktop of vWorkstation.

When transitioning to production, your team would probably be asked to create automation scripts to do much of this deployment work. This could be done using tools like Ansible or Chef. Your job now is to get the basic deployment working. Since the helper scripts are currently on vWorkstation, you need to copy them to cloud1.

10. On the vWorkstation desktop, **double-click** the **Command Prompt shortcut** to open a second command prompt window.

11. In the new command prompt window, **type** `scp "*.sh" cloud1:~/.` and **press Enter** to copy the scripts to cloud1.



```
C:\Users\Administrator\Desktop>scp "*.sh" cloud1:~/.
add_db.sh              100%  227     7.0KB/s   00:00
add_service.sh         100%  566    35.6KB/s   00:00
admin-openrc.sh        100%  230    14.4KB/s   00:00
```

Command to copy scripts

Note: The command prompt shows the working directory as the Desktop folder. The scp command uses the wildcard pattern *.sh to copy all the files with the .sh extension from the Desktop to the location ~/. on cloud1. The ~ is a shortcut for the home directory of the user, /home/user, and the dot after the slash means to use the same name for each file on the remote machine. You should see that three files were copied.

12. **Close** the second command prompt window that was opened.

13. In the command prompt window for cloud1, **type** `ls` and **press Enter** to view the files in /home/user.

Note: You can now see that the three scripts are available on cloud1. However, they are not executable.

14. In the command prompt window for cloud1, **type** `chmod 755 *.sh` and **press Enter** to make the scripts all executable.

Note: Now you can begin the work of actually setting up OpenStack. You will start with the keystone service, which is actually a collection of several services that work together to provide authentication (prove identity) and authorization (grant access based on identity). One part of the service manages the data for the different objects associated with managing identities.

- Domains – A domain is a namespace for the other objects managed by the service. Each user, group, role, or project is owned by exactly one domain.

- Users – A user is an individual consumer of an API in the system. Usernames must be unique within a domain.

- Groups – A group is a collection of one or more users. Group names must be unique within a domain.

- Roles – A role is an object used to manage authorization.

- Projects – A project is the basic unit of ownership within the system. All resources belong to a specific project.

The Identity part of the keystone service manages Users and Groups. The Resource part manages projects and domains. The Assignment service manages data relating to roles and the assignment of roles. Roles are used along with Identities to create a tuple (Role, Resource, Identity) used to manage authorization.

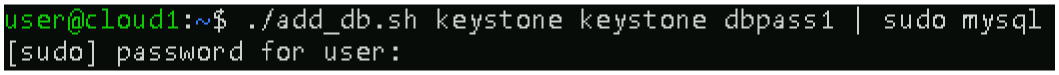
15. On the vWorkstation desktop, **double-click** the **add_db.sh icon** to open the file in Notepad++.

Shortcut to add_db.sh script

Note: Each service stores data in the database on the controller node. This script is used to create a database and a corresponding user for accessing that database. The form of the commands required for each service are very similar and can easily be parameterized. The first line indicates that the script should be run using /bin/sh. Lines 3 through 5 assign the first three command line arguments of the script to variables named DB, USER, and PASSWD. The remaining lines echo some SQL using the passed values. This script is set up to create the SQL commands needed to create a new database on the MariaDB server and add a user that is allowed to connect to that database with the given password.

16. **Close** the **Notepad++ window**.

17. In the command prompt, **type** **`./add_db.sh keystone keystone dbpass1 | sudo mysql`** and **press Enter** to add the keystone database. When prompted for a password, **type** **`password`** and **press Enter**.



```
user@cloud1:~$ ./add_db.sh keystone keystone dbpass1 | sudo mysql
[sudo] password for user:
```

Command to add keystone database

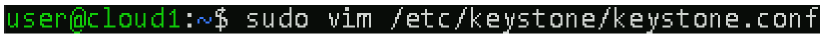
Note: You will be able to type your password into sudo, but it will not make your text visible.

This command works by sending the SQL constructed by the script to the mysql command. The mysql command is a tool for connecting to a MySQL compatible database (like MariaDB). It can be used interactively. It can also be used non-interactively as shown by passing command to the standard input of the command. The use of sudo is required to connect to the database as root to have the privileges required for creating a new database.

This database will be used by the keystone service to hold any information the service needs. You will use a command from the keystone package to initialize the database after a few more steps. At this point, you have only created the database with the name that OpenStack expects and created a database user named *keystone* who can connect to the database using the password *dbpass1*.

Now you need to edit the configuration file for the keystone service so that it can be started. The Ubuntu package has already been installed but is not yet active. The configuration file for the keystone service is at /etc/keystone/keystone.conf.

18. In the command prompt window, **type** `sudo vim /etc/keystone/keystone.conf` and **press Enter** to open the configuration file in an editor. If prompted for a password, **type** `password` and **press Enter**.



```
user@cloud1:~$ sudo vim /etc/keystone/keystone.conf
```

Command to edit keystone configuration

Note: Most of the configuration files that will be edited are not owned by the *user* account. For many, the directory containing the files does not even allow user to navigate them. So, the sudo command must be used frequently for file operations, and tab completion may not work for entering in some of the paths. The file you see in the editor is not the original file from the keystone service package. As is typical with many Linux services, the installed default configuration file is heavily commented and shows many of the options available. However, this obscures the actually small number of settings that need to be managed for this simple deployment scenario. The original configuration file is available in the folder Original Configs on the desktop of vWorkstation.

For editing files, the vim editor is nearly always available in some form on Unix-like systems. Navigation in the Vim editor can be done with the arrow keys or the h (left), j (down), k (up), and l (right) keys. The i key enters insert mode, where text can be added and deleted. The Escape key leaves insert mode.

19. In the editor, **navigate the cursor** to the **s in sqlite** on the line starting with connection.

20. In the editor, **type** `d$` to delete from the cursor to the end of the line.

21. In the editor, **type** `a` to begin appending text to the line.

22. In the editor, **type** `mysql+pymysql://keystone:dbpass1@cloud1/keystone` and **press Escape**.

23. In the editor, **type** `:w` and **press Enter** to save the changes so far.

Note: The connection line tells the keystone service how to connect to the database. It uses an SQLAlchemy string that gives the protocol (mysql using a Python binding), the user *keystone* and password *dbpass1*, the host *cloud1*, and the database name *keystone*.

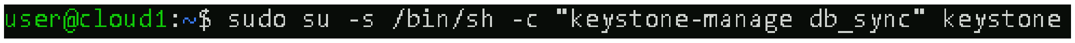
24. In the editor, **navigate the cursor** to the **# at the start of the line with "provider = fernet"** and **type** `x` to uncomment the line.

Note: The provider specifies the type of token used for authorization in the identity service. The fernet token type provides some benefits for web-based applications. You can learn more about this type of token here.

25. In the editor, **type** `:wq` and **press Enter** to save the changes and exit.

Note: Now the configuration for the Keystone service is complete. Although there are a few more things to set up before the service can be started. These steps will all use the keystone-manage command from the keystone package. For later services, some of these similar steps will be automated with the helper scripts.

26. At the command prompt, **type** `sudo su -s /bin/sh -c "keystone-manage db_sync" keystone` and **press Enter** to initialize the database.

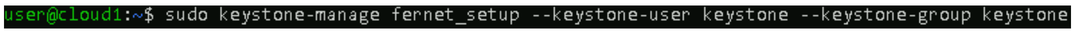


```
user@cloud1:~$ sudo su -s /bin/sh -c "keystone-manage db_sync" keystone
```

Command to initialize the keystone database

Note: This step may take a little bit of time to complete and should produce no output.

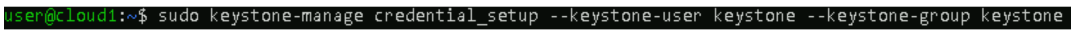
27. At the command prompt, **type** `sudo keystone-manage fernet_setup --keystone-user keystone --keystone-group keystone` and **press Enter.**



```
user@cloud1:~$ sudo keystone-manage fernet_setup --keystone-user keystone --keystone-group keystone
```

Command to initialize token encryption

28. At the command prompt, **type** `sudo keystone-manage credential_setup --keystone-user keystone --keystone-group keystone` and **press Enter**.

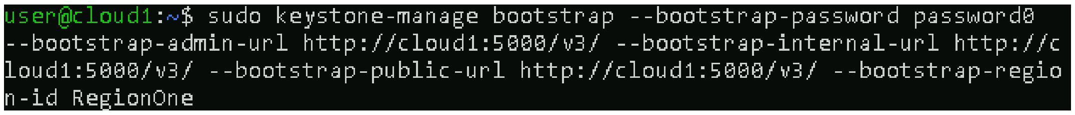


```
user@cloud1:~$ sudo keystone-manage credential_setup --keystone-user keystone --keystone-group keystone
```

Command to initialize credential encryption

Note: These two commands initialize the credential handling. The first creates a fernet key repository for token encryption. The second creates a fernet key repository for credential encryption. There should be no output from them. The final command has quite a few options.

29. At the command prompt, **type** `sudo keystone-manage bootstrap --bootstrap-password password0 --bootstrap-admin-url http://cloud1:5000/v3/ --bootstrap-internal-url http://cloud1:5000/v3/ --bootstrap-public-url http://cloud1:5000/v3/ --bootstrap-region-id RegionOne` and **press Enter**.



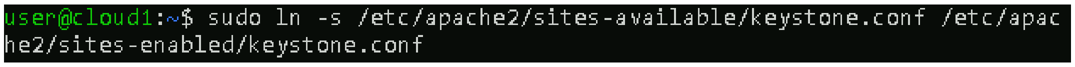
```
user@cloud1:~$ sudo keystone-manage bootstrap --bootstrap-password password0
--bootstrap-admin-url http://cloud1:5000/v3/ --bootstrap-internal-url http://c
loud1:5000/v3/ --bootstrap-public-url http://cloud1:5000/v3/ --bootstrap-regio
n-id RegionOne
```

Command to bootstrap the keystone service

Note: This command bootstraps the keystone service by defining the endpoints for the service that can be used for public interactions, internal interactions, and administrative interactions. All of this is associated with a region, which provides another method of grouping in the OpenStack system. For this lab, everything will be in the same region.

The actual service runs as a Python WSGI service that is proxied by Apache. WSGI is a protocol that allows Apache to interact with an external program that provides web pages. The web server is already installed and running on the cloud1 machine (from the standard Ubuntu Server installation). The keystone package added a file keystone.conf to the /etc/apache2/sites-available directory. Rather than copying the file, you can just create a symbolic link to the configuration from the sites-enabled directory.

30. At the command prompt, **type** `sudo ln -s /etc/apache2/sites-available/keystone.conf /etc/apache2/sites-enabled/keystone.conf` and **press Enter** to create the symbolic link.



```
user@cloud1:~$ sudo ln -s /etc/apache2/sites-available/keystone.conf /etc/apac
he2/sites-enabled/keystone.conf
```

Command to enable the keystone web API

Note: This links the website configuration for keystone so that when the web server is restarted, it will become active. But first, you want to let the web server know what its name is.

31. At the command prompt, **type** `grep ServerName /etc/apache2/apache2.conf` and **press Enter** to search the configuration for a server name.

Note: You should see that the string ServerName does not yet occur in the configuration file.

32. At the command prompt, **type** `sudo vim /etc/apache2/apache2.conf` and **press Enter** to open the configuration file in an editor.

33. **Scroll down** in the editor to the **first blank line after a line with Global configuration**. **Place** the **cursor** at the start of the line.

34. In the editor, **type** `i` to enter insert mode.

35. In the editor, **type** `ServerName cloud1` and **press Escape** to set the server's name.

36. In the editor, **type** `:wq` and **press Enter** to save the file and exit.

Note: Now you can restart the apache service, which should also start the keystone service.

37. At the command prompt, **type** `sudo systemctl restart apache2` and **press Enter** to restart the apache service.

Note: With the keystone service running, the OpenStack Python client can now be used for configuration. This client has many options for telling it where the identity service provider can be found. To avoid lots of very long command lines, it can also use environment variables. One of the scripts that was copied from vWorkstation, admin-openrc.sh, contains settings for the variables that tell the OpenStack client to use the keystone service you just set up.

38. On the vWorkstation desktop, **double-click** the **admin-openrc.sh icon** to open the file in Notepad++.

Shortcut icon for admin-openrc.sh

Note: This script simply sets several environment variables. Then, when OpenStack commands are executed, the values of those variables are used to provide some options.

- OS_AUTH_URL – Specify the URL where the keystone service is listening.
- OS_PROJECT_DOMAIN_NAME – All projects are part of some domain. This variable specifies which domain to use.
- OS_USER_DOMAIN_NAME – Similarly, all users belong to a domain.
- OS_PROJECT_NAME – Specify the name of the project.
- OS_USERNAME and OS_PASSWORD – Specify the user and password to authorize with.

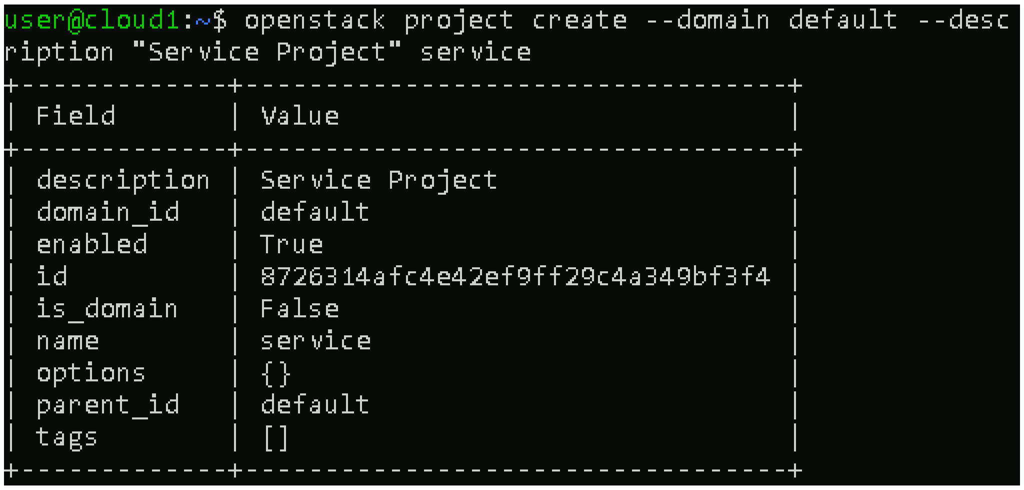
39. **Close Notepad++**.

40. At the command prompt for cloud1, **type** `source admin-openrc.sh` and **press Enter** to set the environment variables.

41. At the command prompt, **type** `env | grep ^OS_` and **press Enter** to view the current environment variables starting with OS_.

Note: You should see all the variables that were defined in the script file.

42. At the command prompt, **type** `openstack project create --domain default --description "Service Project" service` and **press Enter** to create a new project.



```
user@cloud1:~$ openstack project create --domain default --desc
ription "Service Project" service
+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description | Service Project                  |
| domain_id   | default                          |
| enabled     | True                             |
| id          | 8726314afc4e42ef9ff29c4a349bf3f4 |
| is_domain   | False                            |
| name        | service                          |
| options     | {}                               |
| parent_id   | default                          |
| tags        | []                               |
+-------------+----------------------------------+
```

Command to create the service project

Note: This command has a few pieces.

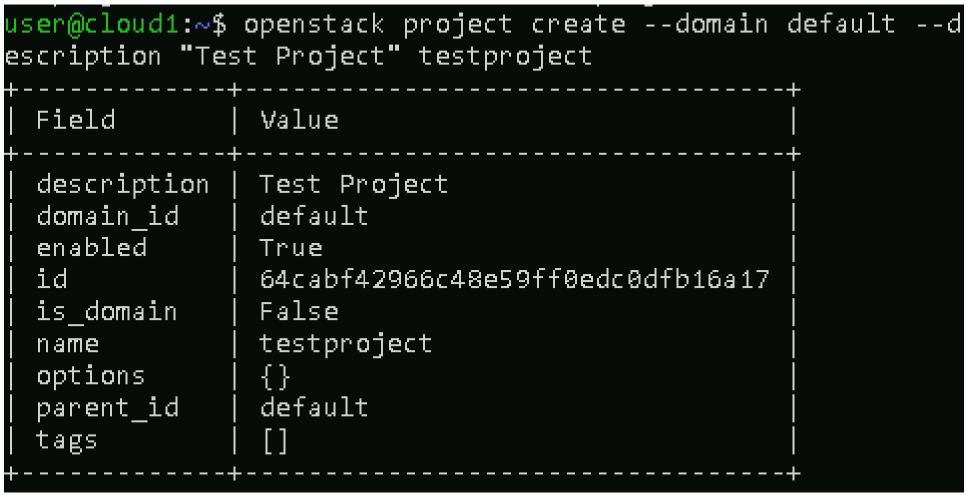
- openstack – this is the Python OpenStack client command
- project – a subcommand to work with projects
- create – a subcommand to create a new project
- --domain default – an option to set the domain for the new project

- --description “Service Project” – an option to associate a human readable description with the project

- service – the name for the project

After running the command (which may take a few seconds to complete), you should see information about the newly created project. You also want a test project and user to be used for testing the system later.

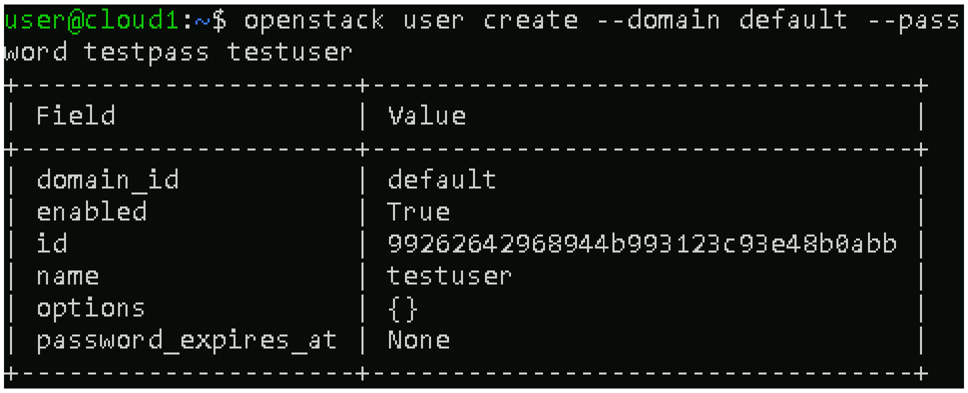
43. At the command prompt, **type** `openstack project create --domain default --description "Test Project" testproject` and **press Enter** to create the test project.



```
user@cloud1:~$ openstack project create --domain default --d
escription "Test Project" testproject
+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description | Test Project                     |
| domain_id   | default                          |
| enabled     | True                             |
| id          | 64cabf42966c48e59ff0edc0dfb16a17 |
| is_domain   | False                            |
| name        | testproject                      |
| options     | {}                               |
| parent_id   | default                          |
| tags        | []                               |
+-------------+----------------------------------+
```

Command to create the test project

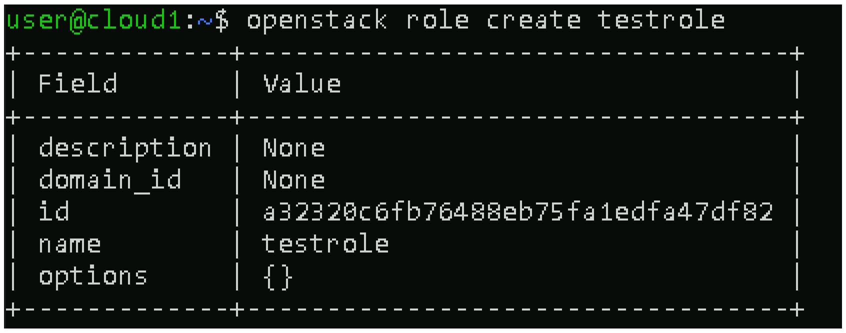
44. At the command prompt, **type** `openstack user create --domain default --password testpass testuser` and **press Enter** to create the test user.



```
user@cloud1:~$ openstack user create --domain default --pass
word testpass testuser
+---------------------+----------------------------------+
| Field               | Value                            |
+---------------------+----------------------------------+
| domain_id           | default                          |
| enabled             | True                             |
| id                  | 99262642968944b993123c93e48b0abb |
| name                | testuser                         |
| options             | {}                               |
| password_expires_at | None                             |
+---------------------+----------------------------------+
```

Command to create a test user

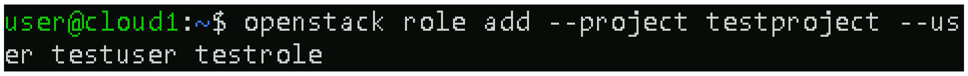
45. At the command prompt, **type** **openstack role create testrole** and **press Enter** to create a test role.



```
user@cloud1:~$ openstack role create testrole
+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description | None                             |
| domain_id   | None                             |
| id          | a32320c6fb76488eb75fa1edfa47df82 |
| name        | testrole                         |
| options     | {}                               |
+-------------+----------------------------------+
```

Command to create a test role

46. At the command prompt, **type** **openstack role add --project testproject --user testuser testrole** and **press Enter** to add the test user to the test project with the test role.



```
user@cloud1:~$ openstack role add --project testproject --us
er testuser testrole
```

Command to add the test user to the test project

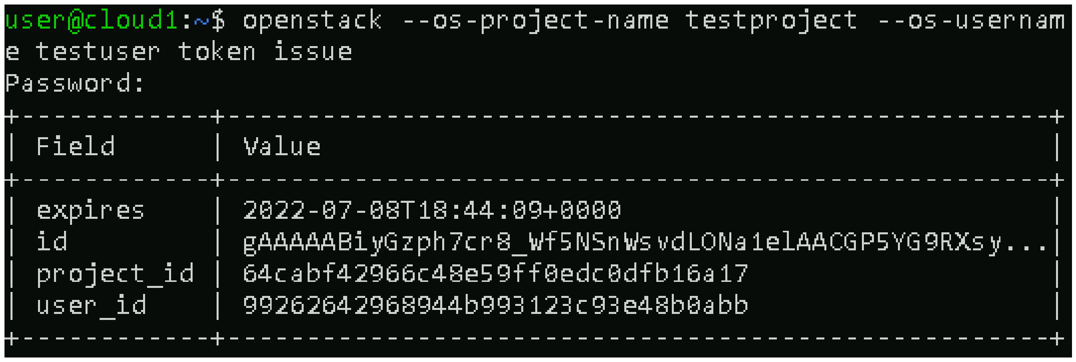
Note: The first three of these commands should complete in a few seconds and show information about the created entity. The final command should have no output.

47. **Make a screen capture** showing the information about the test project, test user, and test role.

Note: As a final test of the keystone service, you can use the openstack command to explicitly create an authorization token.

48. At the command prompt, **type** **`unset OS_PASSWORD`** and **press Enter** to remove the environment variable for the password.

49. At the command prompt, **type** **`openstack --os-project-name testproject --os-username testuser token issue`** and **press Enter** to request a token.



```
user@cloud1:~$ openstack --os-project-name testproject --os-usernam
e testuser token issue
Password:
+------------+-------------------------------------------------+
| Field      | Value                                           |
+------------+-------------------------------------------------+
| expires    | 2022-07-08T18:44:09+0000                        |
| id         | gAAAAABiyGzph7cr8_Wf5NSnWsvdLONa1elAACGP5YG9RXsy...|
| project_id | 64cabf42966c48e59ff0edc0dfb16a17                |
| user_id    | 99262642968944b993123c93e48b0abb                |
+------------+-------------------------------------------------+
```

Command to issue a token

Note: Here you have used some command line options –os-* to override some of the other environment variable values. Since OS_PASSWORD was unset, this command will prompt for a password.

50. When prompted, **type** `testpass` and **press Enter** to give the password for testuser.

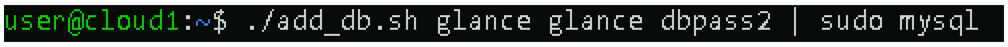
Note: You will be able to type your password, but your typed characters will not be displayed on the screen.
You should see output giving a token from the keystone service. This token can be used to perform actions that testuser is authorized to do in the OpenStack cloud that is being set up.

51. **Make a screen capture** showing the generated token information.

52. At the command prompt, **type** `source admin-openrc.sh` and **press Enter** to reset the environment.

Note: Now you will install two other services that are essential but not very complicated. The glance service is used to manage images for virtual machines. In the final deployment, this configuration will be updated to use the “cinder” block storage service for storing images. For your testing, you will just store the images in files on the controller node.

53. At the command prompt, **type** `./add_db.sh glance glance dbpass2 | sudo mysql` and **press Enter** to create the database for the service.



```
user@cloud1:~$ ./add_db.sh glance glance dbpass2 | sudo mysql
```

Command to add the glance database

Note: This creates a new database that can be accessed by the user glance with the password dbpass2. You will now make use of the add_service.sh script.

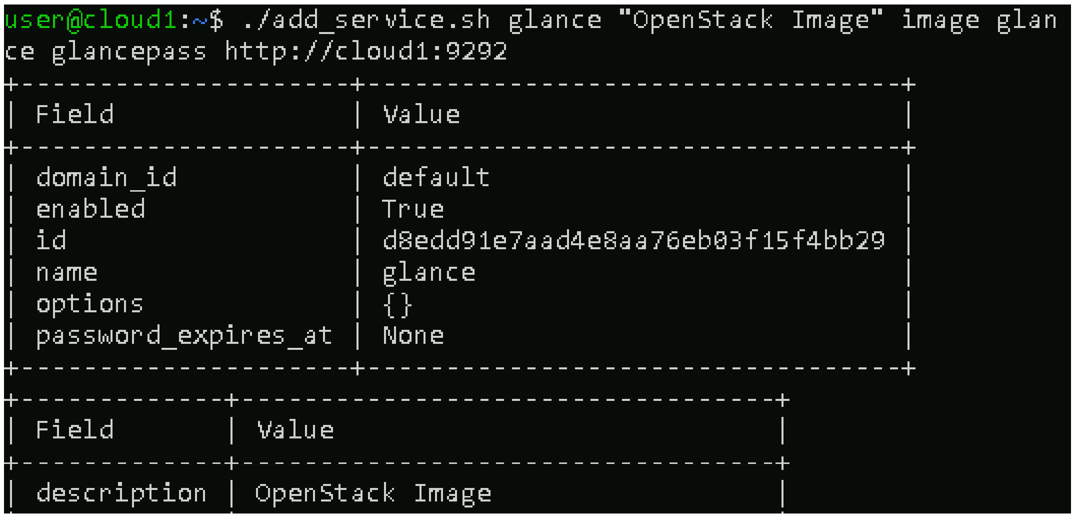
54. On the vWorkstation desktop, **double-click** on the **add_service.sh icon** to open the script in Notepad++.

Shortcut to the add_service.sh script

Note: This script takes six arguments. It uses the openstack command to add a user for the service, give that user admin rights in the service project, create a new service, and then define the public, internal, and admin endpoints for the service. These commands will need to be used multiple times for each service you add, so using a script makes it a bit easier.

55. **Close** the **Notepad++ window**.

56. At the command prompt, **type** `./add_service.sh glance "OpenStack Image" image glance glancepass http://cloud1:9292` and **press Enter** to set up the glance service.



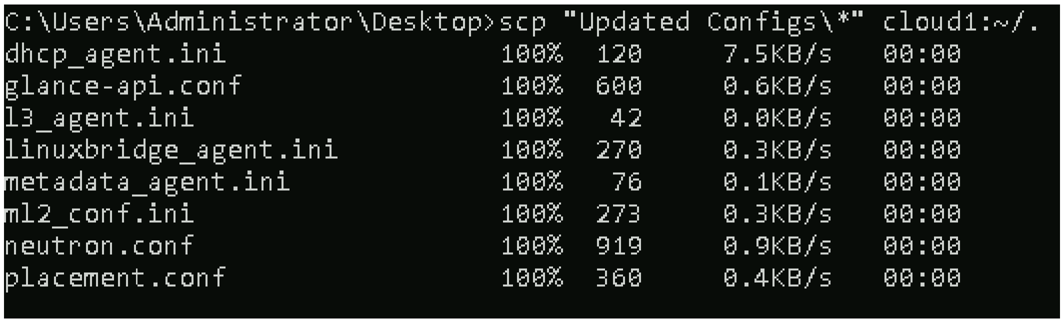
```
user@cloud1:~$ ./add_service.sh glance "OpenStack Image" image glan
ce glancepass http://cloud1:9292
+---------------------+----------------------------------+
| Field               | Value                            |
+---------------------+----------------------------------+
| domain_id           | default                          |
| enabled             | True                             |
| id                  | d8edd91e7aad4e8aa76eb03f15f4bb29 |
| name                | glance                           |
| options             | {}                               |
| password_expires_at | None                             |
+---------------------+----------------------------------+
+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description | OpenStack Image                  |
```

Command to setup the glance service

Note: You should see the output from each of the commands all together after the script completes. The next step is configuring the service. As with the keystone configuration, the original files are in Original Configs on vWorkstation. There is also a folder named Updated Configs where some of the configuration files have already been updated by your team.

57. On the vWorkstation desktop, **double-click** the **Command Prompt shortcut** to open a second command prompt window.

58. At the new command prompt, **type** `scp "Updated Configs\*" cloud1:~/.` and **press Enter** to copy all the configuration files to the /home/user directory on cloud1.

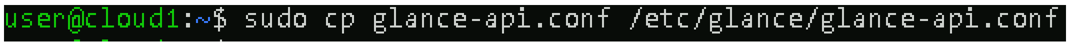


```
C:\Users\Administrator\Desktop>scp "Updated Configs\*" cloud1:~/.
dhcp_agent.ini                100%  120     7.5KB/s   00:00
glance-api.conf               100%  600     0.6KB/s   00:00
l3_agent.ini                  100%   42     0.0KB/s   00:00
linuxbridge_agent.ini         100%  270     0.3KB/s   00:00
metadata_agent.ini            100%   76     0.1KB/s   00:00
ml2_conf.ini                  100%  273     0.3KB/s   00:00
neutron.conf                  100%  919     0.9KB/s   00:00
placement.conf                100%  360     0.4KB/s   00:00
```

Command to copy the configuration files

59. **Close** the **second command prompt window**.

60. At the original command prompt for cloud1, **type** `sudo cp glance-api.conf /etc/glance/glance-api.conf` and **press Enter** to copy the updated configuration to its proper location.

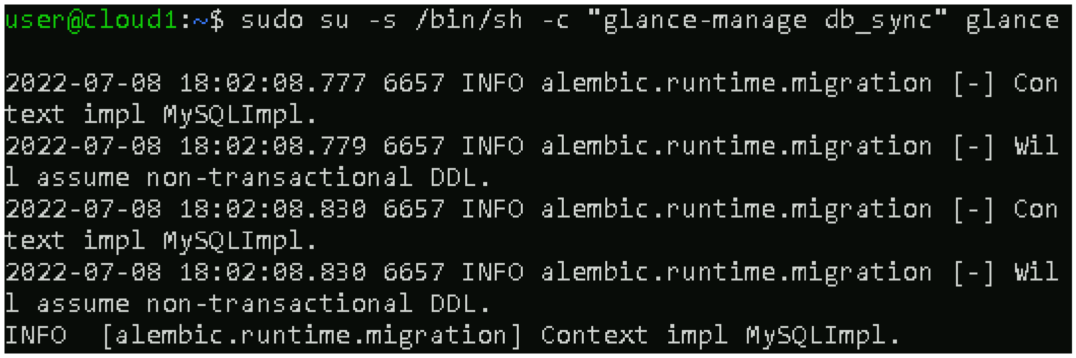


```
user@cloud1:~$ sudo cp glance-api.conf /etc/glance/glance-api.conf
```

Command to install the glance configuration file

Note: Now you can initialize the database and start the glance service. This service runs as a standard Linux service and is controlled with systemctl.

61. At the command prompt, **type** **`sudo su -s /bin/sh -c "glance-manage db_sync" glance`** and p**ress Enter** to initialize the database for the service.

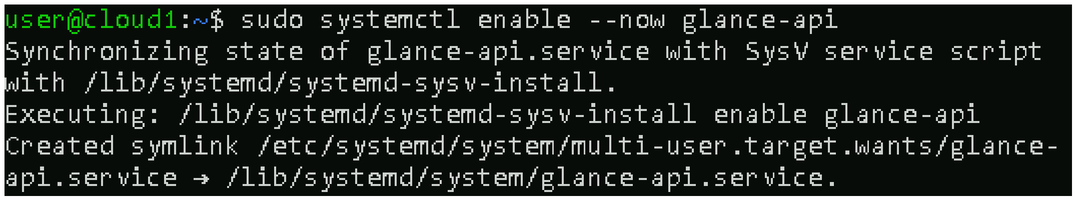


```
user@cloud1:~$ sudo su -s /bin/sh -c "glance-manage db_sync" glance

2022-07-08 18:02:08.777 6657 INFO alembic.runtime.migration [-] Con
text impl MySQLImpl.
2022-07-08 18:02:08.779 6657 INFO alembic.runtime.migration [-] Wil
l assume non-transactional DDL.
2022-07-08 18:02:08.830 6657 INFO alembic.runtime.migration [-] Con
text impl MySQLImpl.
2022-07-08 18:02:08.830 6657 INFO alembic.runtime.migration [-] Wil
l assume non-transactional DDL.
INFO  [alembic.runtime.migration] Context impl MySQLImpl.
```

Command to initialize the glance database

62. At the command prompt, **type** **`sudo systemctl enable --now glance-api`** and **press Enter** to enable and start the service.

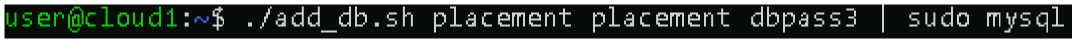


```
user@cloud1:~$ sudo systemctl enable --now glance-api
Synchronizing state of glance-api.service with SysV service script
with /lib/systemd/systemd-sysv-install.
Executing: /lib/systemd/systemd-sysv-install enable glance-api
Created symlink /etc/systemd/system/multi-user.target.wants/glance-
api.service → /lib/systemd/system/glance-api.service.
```

Command to enable the glance service

Note: The final service to be enabled in this part is the placement service. The commands are all very similar to the steps that have already been used. This service also runs behind apache.

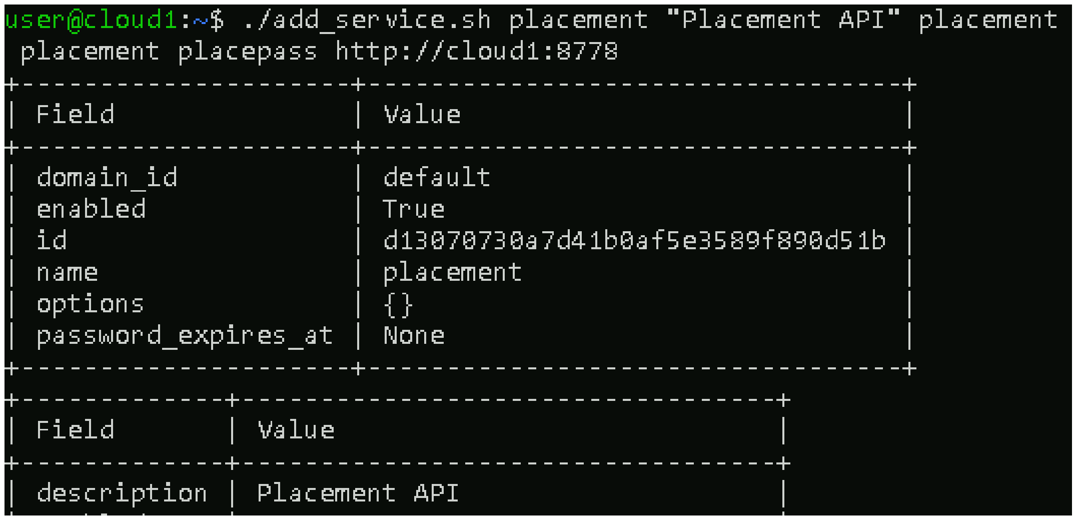
63. At the command prompt, **type** **`./add_db.sh placement placement dbpass3 | sudo mysql`** and **press Enter** to create the placement database.



```
user@cloud1:~$ ./add_db.sh placement placement dbpass3 | sudo mysql
```

Command to add the placement database

64. At the command prompt, **type** **`./add_service.sh placement "Placement API" placement placement placepass http://cloud1:8778`** and **press Enter** to register the service.

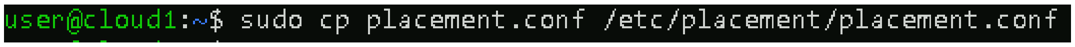


```
user@cloud1:~$ ./add_service.sh placement "Placement API" placement
 placement placepass http://cloud1:8778
+---------------------+----------------------------------+
| Field               | Value                            |
+---------------------+----------------------------------+
| domain_id           | default                          |
| enabled             | True                             |
| id                  | d13070730a7d41b0af5e3589f890d51b |
| name                | placement                        |
| options             | {}                               |
| password_expires_at | None                             |
+---------------------+----------------------------------+
+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description | Placement API                    |
```

Command to register the placement service

Note: As with the glance service, first you create the database and register the service.

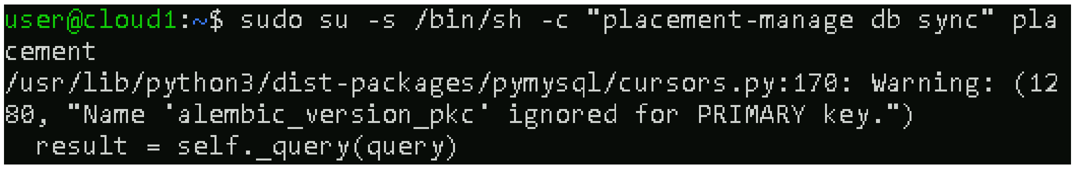
65. At the command prompt, **type** **`sudo cp placement.conf /etc/placement/placement.conf`** and **press Enter** to update the configuration file.



```
user@cloud1:~$ sudo cp placement.conf /etc/placement/placement.conf
```

Command to install the placement configuration file

66. At the command prompt, **type** **`sudo su -s /bin/sh -c "placement-manage db sync" placement`** and **press Enter** to initialize the database.

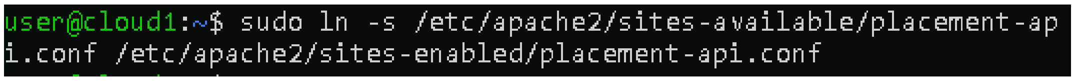


```
user@cloud1:~$ sudo su -s /bin/sh -c "placement-manage db sync" pla
cement
/usr/lib/python3/dist-packages/pymysql/cursors.py:170: Warning: (12
80, "Name 'alembic_version_pkc' ignored for PRIMARY key.")
  result = self._query(query)
```

Command to initialize the placement database

Note: Then you update the configuration and initialize the database.

67. At the command prompt, **type** **`sudo ln -s /etc/apache2/sites-available/placement-api.conf /etc/apache2/sites-enabled/placement-api.conf`** and **press Enter** to link the apache configuration.



```
user@cloud1:~$ sudo ln -s /etc/apache2/sites-available/placement-ap
i.conf /etc/apache2/sites-enabled/placement-api.conf
```

Command to enable the placement API

Note: Finally, restart the apache web server to realize your changes.

68. At the command prompt, **type** `sudo systemctl restart apache2` and **press Enter** to restart Apache.

Note: Now the keystone, glance, and placement services are all running on the controller node cloud1.

Part 2: Set Up the Compute Service

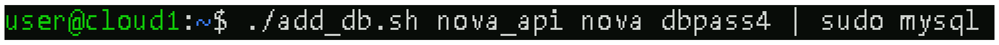
Note: The compute service is an integral part of the OpenStack system. It provides management of the virtual machine instances. As with the keystone service, the compute service (nova) is actually constructed from multiple smaller services. In current versions of OpenStack, the nova architecture uses the Cell (v2) setup. Cells provide a mechanism for grouping clusters of compute nodes and configuring them together. The nova service provides APIs for managing instances.

In a simple small deployment, there are two cells. Cell 0 is reserved to contain instances that failed to start, so it does not actually contain any running instances. All the actual instances will be placed in cell 1. This is sometimes called a single cell deployment. The nova system with just these two cells has four major parts that all communicate via the message queue service.

- nova-api – provides the high-level API interactions
- nova-conductor – takes over long-running tasks for nova-api and talks to the databases for nova-compute
- nova-scheduler – tracks resources and figures out where to put instances
- nova-compute – manages the virtualization of the instances

If more cells are needed, then each new cell gets its own nova-conductor and nova-compute instance as well as a message broker. A new component called the “super conductor” manages interactions between the API level requests and the cells.
For the test deployment, your team has decided that the simple “single cell” deployment is sufficient. The nova service uses the networking service for communication, so the configuration will also involve a section specifying how to talk to the networking service. First, you will install the controller node portion of the nova service on cloud1. You will begin with the usual step of creating the database. In this case, you create three.

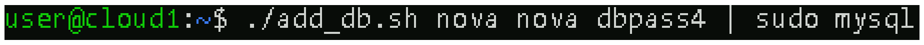
1. At the command prompt for cloud1, **type** `./add_db.sh nova_api nova dbpass4 | sudo mysql` and **press Enter** to create the nova_api database.



```
user@cloud1:~$ ./add_db.sh nova_api nova dbpass4 | sudo mysql
```

Command to add the nova_api database

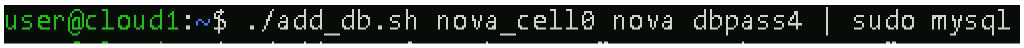
2. At the command prompt, **type** `./add_db.sh nova nova dbpass4 | sudo mysql` and **press Enter** to create the nova database.



```
user@cloud1:~$ ./add_db.sh nova nova dbpass4 | sudo mysql
```

Command to add the nova database

3. At the command prompt, **type** `./add_db.sh nova_cell0 nova dbpass4 | sudo mysql` and **press Enter** to create the nova_cell0 database.

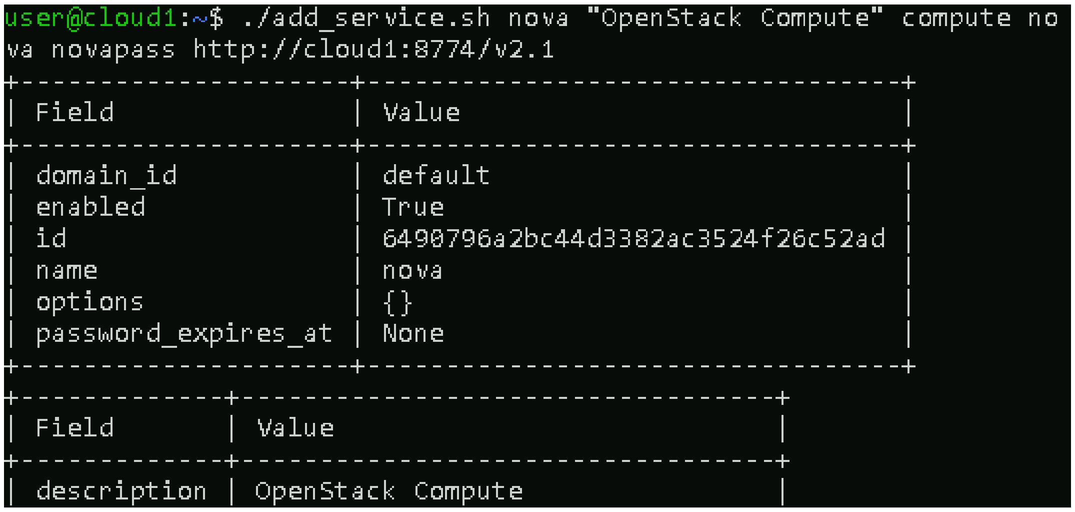


```
user@cloud1:~$ ./add_db.sh nova_cell0 nova dbpass4 | sudo mysql
```

Command to add the nova_cell0 database

Note: Then you register the service (only one here).

4. At the command prompt, **type** `./add_service.sh nova "OpenStack Compute" compute nova novapass http://cloud1:8774/v2.1` and **press Enter** to register the nova service.



```
user@cloud1:~$ ./add_service.sh nova "OpenStack Compute" compute no
va novapass http://cloud1:8774/v2.1
+---------------------+----------------------------------+
| Field               | Value                            |
+---------------------+----------------------------------+
| domain_id           | default                          |
| enabled             | True                             |
| id                  | 6490796a2bc44d3382ac3524f26c52ad |
| name                | nova                             |
| options             | {}                               |
| password_expires_at | None                             |
+---------------------+----------------------------------+
+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description | OpenStack Compute                |
```

Command to register the nova service

5. At the command prompt, **type** `sudo vim /etc/nova/nova.conf` and **press Enter** to open the configuration file.

Note: The configuration for the nova service has multiple sections that need to be updated. This is because nova talks to various other services within OpenStack, so it needs to know how to communicate with them. Refer back to Part 1, steps 20–26 for instructions on how to delete, append, save, uncomment, and save and exit in the editor. Make sure you uncomment the lines as you go.

6. In the editor, **add two new lines** in the **[DEFAULT] section**

```
transport_url = rabbit://openstack:rabbitpass@cloud1:5672/
my_ip = 172.30.0.10
```

Note: The transport URL tells the service how to talk to the message queue service. The username and password were chosen by the team member who set up the original services, in this case *openstack* and *rabbitpass*. The service also needs to talk to the nova-api and nova databases. These use the familiar SQLAlchemy URIs.

7. In the editor, **update** the **connection in the [api_database] section** to the following: `mysql+pymysql://nova:dbpass4@cloud1/nova_api`

8. In the editor, **update** the **connection line in the [database] section** to the following: `mysql+pymysql://nova:dbpass4@cloud1/nova`

9. In the editor, **update** the **lock_path in the [oslo_concurrency] section** to the following: `/var/lib/nova/tmp`.

Note: Some of the system operations require locking among multiple separate processes. This setting specifies a prefix where files used for the locking mechanism are stored. Now you will update the information about the keystone service.

10. In the editor, **uncomment** the **auth strategy line** in the [api] section.

11. In the editor, **update the [keystone_authtoken] section** so that it has the following lines:

```
www_authenticate_uri = http://cloud1:5000/
auth_url = http://cloud1:5000/
memcached_servers = cloud1:11211
auth_type = password
project_domain_name = Default
user_domain_name = Default
project_name = service
username = nova
password = novapass
```

Note: These settings are familiar from the environment variables used on the command line. They specify how to connect to the keystone service and give the username and password that should be used by the nova service.

12. In the editor, **update the [glance] section** so that it has the following line: `api_servers = http://cloud1:9292`

Note: Since the machines are built based on images, the nova service needs to communicate with the glance service. This setting tells it where to look. If you compare the url with the settings used with the add_service.sh script for glance, you will see that they are the same.

13. In the editor, **update the [placement] section** so that it has the following lines:

```
auth_type = password
auth_url = http://cloud1:5000/v3
project_name = service
project_domain_name = Default
username = placement
user_domain_name = Default
password = placepass
region_name = RegionOne
```

Note: Here you inform the nova service of how to talk to the placement service.

14. In the editor, **update the [neutron] section** so that it has the following lines:

```
service_metadata_proxy = true
metadata_proxy_shared_secret = secret
auth_type = password
auth_url = http://cloud1:5000
project_name = service
project_domain_name = Default
username = neutron
user_domain_name = Default
password = neutronpass
region_name = RegionOne
```

Note: This group of settings tells nova how to talk to the neutron service that will be configured next. For the final section, the nova service allows external protocols to connect to running instances in the private cloud. One of these protocols is VNC. You will also configure the settings for this.

15. In the editor, **update the [vnc] section** so that it has the following lines:

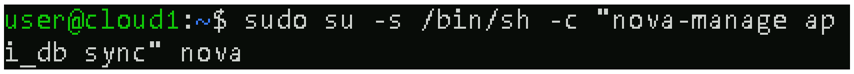
```
enabled = true
server_listen = $my_ip
```

```
server_proxyclient_address = $my_ip
```

16. **Press Escape** and then save and close the file (:wq).

Note: Here you can see the use of the variable my_ip to avoid repeating the same value in multiple places. Now that the configuration is done, you can move on to initializing the databases (all three of them).

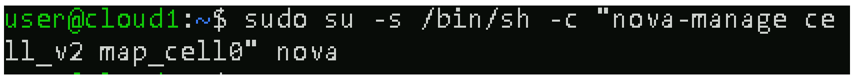
17. At the command prompt, **type** `sudo su -s /bin/sh -c "nova-manage api_db sync" nova` and **press Enter** to initialize the nova api database.



```
user@cloud1:~$ sudo su -s /bin/sh -c "nova-manage ap
i_db sync" nova
```

Command to intialized the nova_api database

18. At the command prompt, **type** `sudo su -s /bin/sh -c "nova-manage cell_v2 map_cell0" nova` and **press Enter**.

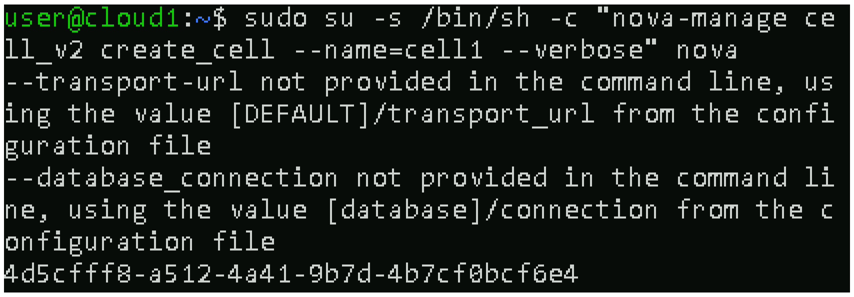


```
user@cloud1:~$ sudo su -s /bin/sh -c "nova-manage ce
ll_v2 map_cell0" nova
```

Command to initialize the cell 0 database

Note: This command initializes the connection to the database for cell 0.

19. At the command prompt, **type** `sudo su -s /bin/sh -c "nova-manage cell_v2 create_cell --name=cell1 --verbose" nova` and **press Enter**.

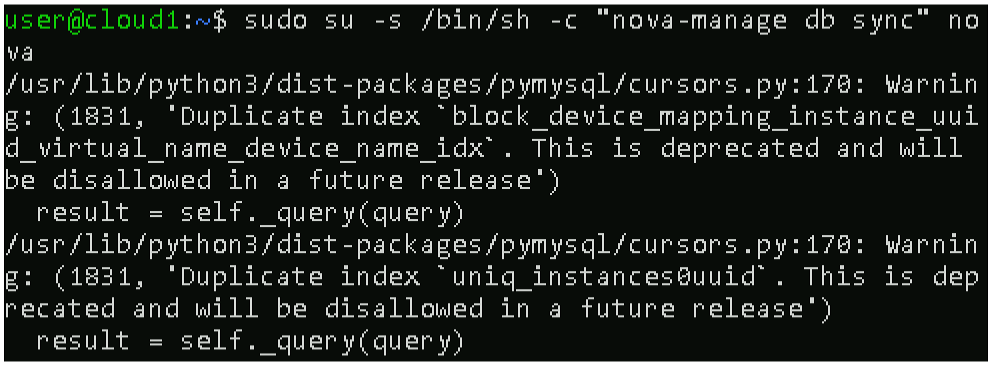


```
user@cloud1:~$ sudo su -s /bin/sh -c "nova-manage ce
ll_v2 create_cell --name=cell1 --verbose" nova
--transport-url not provided in the command line, us
ing the value [DEFAULT]/transport_url from the confi
guration file
--database_connection not provided in the command li
ne, using the value [database]/connection from the c
onfiguration file
4d5cfff8-a512-4a41-9b7d-4b7cf0bcf6e4
```

Command to initialize the cell 1 database

Note: This command initializes the connection for the cell 1 database. You may see some warnings regarding where particular values were retrieved from, which can be safely ignored.

20. At the command prompt, **type** `sudo su -s /bin/sh -c "nova-manage db sync" nova` and **press Enter** to initialize the nova database.

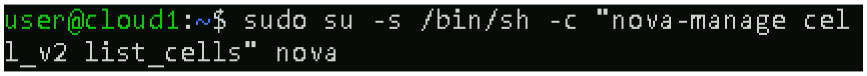


```
user@cloud1:~$ sudo su -s /bin/sh -c "nova-manage db sync" no
va
/usr/lib/python3/dist-packages/pymysql/cursors.py:170: Warnin
g: (1831, 'Duplicate index `block_device_mapping_instance_uui
d_virtual_name_device_name_idx`. This is deprecated and will
be disallowed in a future release')
  result = self._query(query)
/usr/lib/python3/dist-packages/pymysql/cursors.py:170: Warnin
g: (1831, 'Duplicate index `uniq_instances0uuid`. This is dep
recated and will be disallowed in a future release')
  result = self._query(query)
```

Command to initialize the nova database

Note: Now that those databases are all initialized, you can list the cells you have created.

21. At the command prompt, **type** `sudo su -s /bin/sh -c "nova-manage cell_v2 list_cells" nova` and **press Enter** to list the current cells.



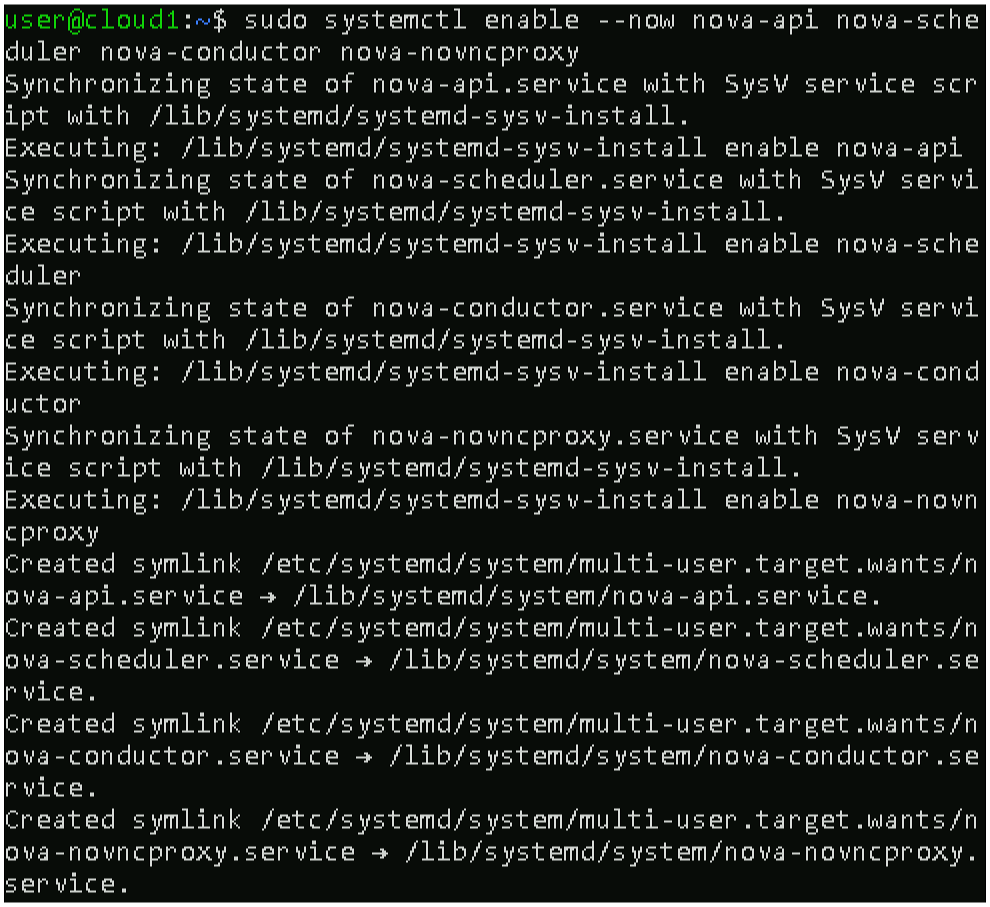
```
user@cloud1:~$ sudo su -s /bin/sh -c "nova-manage cel
l_v2 list_cells" nova
```

Command to list the cells

Note: Both cell0 and cell1 should be represented in the output, showing your successfully configured single cell deployment. Now that all the pieces are configured and initialized for the nova services, it is time to start them, which you will do after the following step.

22. **Make a screen capture** showing cell0 and cell1 in the command output.

23. At the command prompt, **type** `sudo systemctl enable --now nova-api nova-scheduler nova-conductor nova-novncproxy` and **press Enter** to start the nova services.



```
user@cloud1:~$ sudo systemctl enable --now nova-api nova-sche
duler nova-conductor nova-novncproxy
Synchronizing state of nova-api.service with SysV service scr
ipt with /lib/systemd/systemd-sysv-install.
Executing: /lib/systemd/systemd-sysv-install enable nova-api
Synchronizing state of nova-scheduler.service with SysV servi
ce script with /lib/systemd/systemd-sysv-install.
Executing: /lib/systemd/systemd-sysv-install enable nova-sche
duler
Synchronizing state of nova-conductor.service with SysV servi
ce script with /lib/systemd/systemd-sysv-install.
Executing: /lib/systemd/systemd-sysv-install enable nova-cond
uctor
Synchronizing state of nova-novncproxy.service with SysV serv
ice script with /lib/systemd/systemd-sysv-install.
Executing: /lib/systemd/systemd-sysv-install enable nova-novn
cproxy
Created symlink /etc/systemd/system/multi-user.target.wants/n
ova-api.service → /lib/systemd/system/nova-api.service.
Created symlink /etc/systemd/system/multi-user.target.wants/n
ova-scheduler.service → /lib/systemd/system/nova-scheduler.se
rvice.
Created symlink /etc/systemd/system/multi-user.target.wants/n
ova-conductor.service → /lib/systemd/system/nova-conductor.se
rvice.
Created symlink /etc/systemd/system/multi-user.target.wants/n
ova-novncproxy.service → /lib/systemd/system/nova-novncproxy.
service.
```

Command to enable the nova services

Note: That completes the compute services on the controller node. However, you also need to configure the compute services on any compute nodes. You will begin by making one compute node on cloud2. For networking reasons, the compute node should also have two NICs. However, it does not need to have any other support services like databases or message brokers running.

24. On the vWorkstation desktop, **double-click** the **Command Prompt shortcut** to open a new command prompt window.

25. At the new command prompt, **type** `ssh cloud2` and **press Enter** to connect to cloud2.

Note: You should be automatically connected without a password. You will do a quick review of the configuration of this machine.

26. At the command prompt for cloud2, **type** `ip a` and **press Enter** to view the network configuration.

Note: You should see a similar result as you did for cloud1, but with a different IP address. One ethernet interface (ens192) is assigned an IP address of 172.30.0.20 and will be used for the management network. The other ethernet interface (ens224) is not assigned an address and will be used for the provider network.

27. At the command prompt for cloud2, **type** `cat /etc/hosts` and **press Enter** to view the host's file.

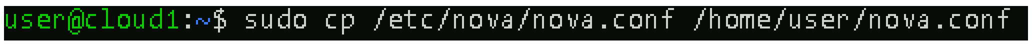
Note: You should see the same contents as on cloud1. Each of the cloud computers is named with its static IP address.

28. At the command prompt for cloud2, **type** `ping -c 5 cloud1` and **press Enter**.

29. At the command prompt for cloud2, **type** `ping -c 5 cloud3` and **press Enter**.

Note: These two commands should succeed and should show five successful packets to the other machines. To configure a compute node, only the nova service needs to be started. One of the configuration files will be nearly the same as the one on cloud1, so you will copy the file over.

30. At the command prompt for cloud1, **type** `sudo cp /etc/nova/nova.conf /home/user/nova.conf` and **press Enter** to copy the nova configuration to the user's home folder.



```
user@cloud1:~$ sudo cp /etc/nova/nova.conf /home/user/nova.conf
```

Command to install the nova configuration file

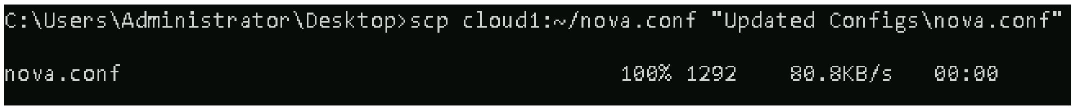
31. At the command prompt for cloud1, **type** `sudo chown user.user nova.conf` and **press Enter** to change the file ownership.

Note: The original file is not readable by user, so the copy was made so that the scp will work.

The setup for the compute node consists of only two configuration files: nova.conf and nova-compute.conf. Most of nova.conf is the same as on the controller, so you will use that as a base.

32. On the vWorkstation desktop, **double-click** the **Command Prompt** shortcut to open a new window.

33. At the new command prompt, **type** `scp cloud1:~/nova.conf "Updated Configs\nova.conf"` and **press Enter** to copy the nova configuration from cloud1 to vWorkstation.

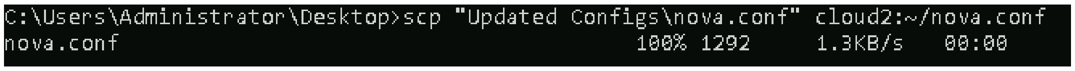


```
C:\Users\Administrator\Desktop>scp cloud1:~/nova.conf "Updated Configs\nova.conf"

nova.conf                                     100% 1292    80.8KB/s   00:00
```

Command to copy the nova configuration to vWorkstation

34. At the same command prompt, **type** `scp "Updated Configs\nova.conf" cloud2:~/nova.conf` and **press Enter** to copy the file to cloud2.



```
C:\Users\Administrator\Desktop>scp "Updated Configs\nova.conf" cloud2:~/nova.conf
nova.conf                                     100% 1292     1.3KB/s   00:00
```

Command to copy the nova configuration to cloud2

35. **Close** the **vWorkstation desktop command prompt window**.

Note: Now you can edit the file on cloud2 to reflect the configuration there.

36. At the command prompt for cloud2, **type** **`vim nova.conf`** and **press Enter** to edit the file.

37. In the editor, **change my_ip** in the **[DEFAULT] section** to **`172.30.0.20`**.

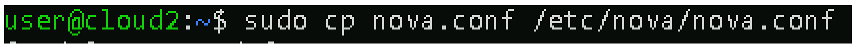
38. In the editor, **remove** the **[database]**, **[api_database], and [neutron]** sections.

Note: The nova compute service does not talk directly to the databases or neutron. Instead, it uses the message broker to communicate with other parts of the nova system.

39. In the editor, **change server_listen** in the **[vnc] section** to **`0.0.0.0`** and **add a new line** **`novncproxy_base_url = http://cloud1:6080/vnc_auto.html`**.

40. In the editor, **type** **`:wq`** and **press Enter** to save and close the file.

41. At the command prompt for cloud2, **type** **`sudo cp nova.conf /etc/nova/nova.conf`** and **press Enter** to copy the updated configuration file to its final location.

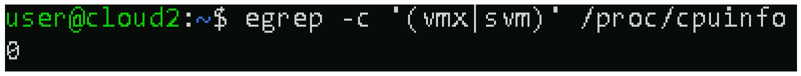


```
user@cloud2:~$ sudo cp nova.conf /etc/nova/nova.conf
```

Command to install the nova configuration file

Note: There is a second configuration file for the compute nodes that needs to be updated. The compute nodes use virtualization to host machine instances. The type of virtualization that can be used depends on the system. If the machine supports hardware acceleration, then the speed can be improved by configuring the nova service to use it.

42. At the command prompt for cloud2, **type** `egrep -c '(vmx|svm)' /proc/cpuinfo` and **press Enter** to check for hardware acceleration.



```
user@cloud2:~$ egrep -c '(vmx|svm)' /proc/cpuinfo
0
```

Command to check the cpu information

Note: This command checks the contents of the /proc/cpuinfo file, which provides information about the machine. If the file contained the strings vmx or svm, then some hardware acceleration could be used. However, this command should return 0 on cloud2, indicating that hardware acceleration is not available.

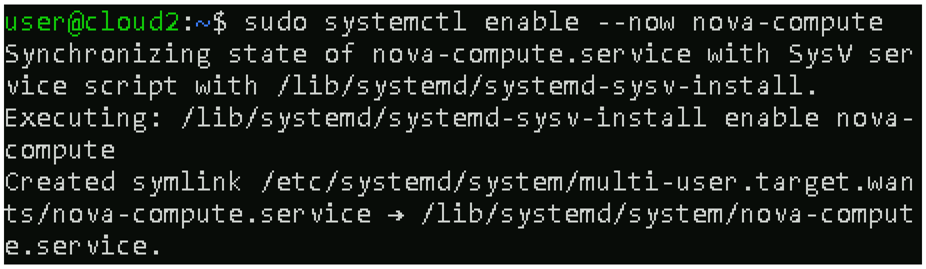
43. At the command prompt for cloud2, **type** `sudo vim /etc/nova/nova-compute.conf` and **press Enter** to edit the second configuration file.

44. In the editor, **change virt_type to** `qemu` in the **[libvirt] section**.

45. In the editor, **type** `:wq` and **press Enter** to save and close the file.

Note: qemu is a method of emulating many types of hardware. The virtualization is not hardware accelerated though, so it tends to be slower. Now you can start the nova compute service on the compute node.

46. At the command prompt for cloud2, **type** `sudo systemctl enable --now nova-compute` and **press Enter**.



```
user@cloud2:~$ sudo systemctl enable --now nova-compute
Synchronizing state of nova-compute.service with SysV ser
vice script with /lib/systemd/systemd-sysv-install.
Executing: /lib/systemd/systemd-sysv-install enable nova-
compute
Created symlink /etc/systemd/system/multi-user.target.wan
ts/nova-compute.service → /lib/systemd/system/nova-comput
e.service.
```

Command to enable the nova compute service

47. **Close** the **command prompt** for cloud2.

Part 3: Set Up the Network Service and Web GUI

Note: The next major piece is the networking service, neutron. The full scope of networking possibilities in OpenStack cannot be covered in a single lab. Your team has decided to start with a slightly less than minimal configuration for the initial networking setup.

The interfaces on the machines that already have IP addresses assigned are going to be the management network. The OpenStack components may communicate over this network, and the network configuration is managed outside the OpenStack system.

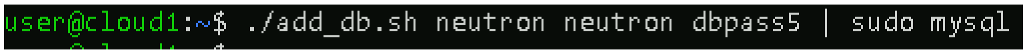
The second interface on each machine is required to create a provider network. A provider network is a network that can connect the virtual machines in the OpenStack cloud to the physical network connected to the machine hosting the cloud. This is a layer-2 connectivity solution, which is why no IP addresses should be assigned to the interfaces.

It is also possible to create completely virtual network components within the OpenStack system. This includes managing subnets, creating routers, and DHCP servers. As noted, the full scope of the options is very broad. For this lab, you will start with enabling the provider network.

You will go back to cloud1 for this. The first few steps are familiar:

1. At the command prompt for cloud1, **type**

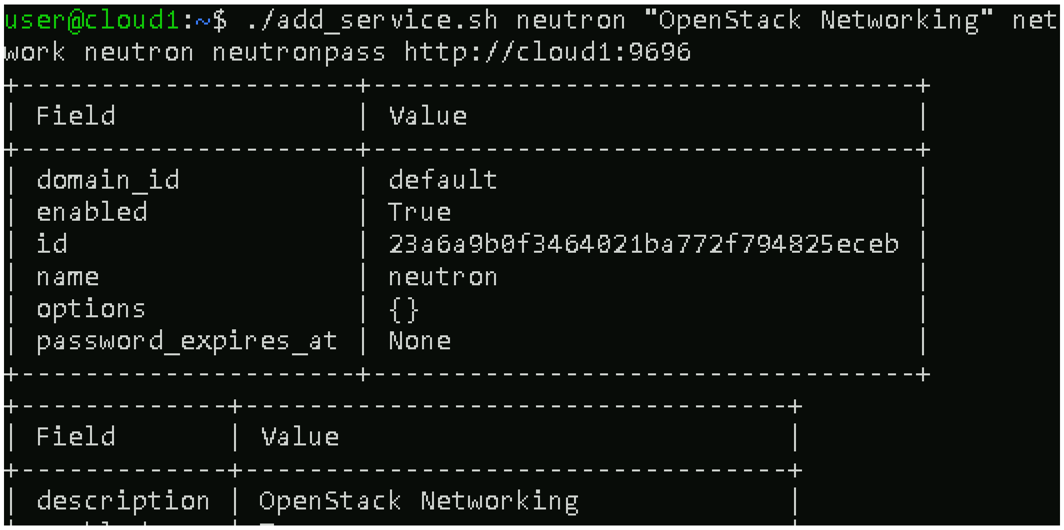
`./add_db.sh neutron neutron dbpass5 | sudo mysql` and **press Enter** to create the database.



```
user@cloud1:~$ ./add_db.sh neutron neutron dbpass5 | sudo mysql
```

Command to add the neutron database

2. At the command prompt for cloud1, **type** `./add_service.sh neutron "OpenStack Networking" network neutron neutronpass http://cloud1:9696` and **press Enter** to register the service.



```
user@cloud1:~$ ./add_service.sh neutron "OpenStack Networking" net
work neutron neutronpass http://cloud1:9696
+---------------------+-----------------------------------+
| Field               | Value                             |
+---------------------+-----------------------------------+
| domain_id           | default                           |
| enabled             | True                              |
| id                  | 23a6a9b0f3464021ba772f794825eceb |
| name                | neutron                           |
| options             | {}                                |
| password_expires_at | None                              |
+---------------------+-----------------------------------+
+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description | OpenStack Networking             |
```

Command to register the neutron service

Note: The networking is provided by another service, which needs its own database and endpoint. You can use the helper scripts again to set these up. A separate member of your team has already begun researching the networking systems and has updated the configuration files for the initial deployment. Most of the changes in neutron.conf are the familiar ones to specify how to talk to other services. There are a few in the DEFAULT section that are new.

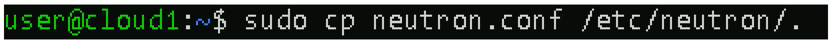
- core_plugin = ml2

- service_plugins = router

- allow_overlapping_ips = true

These are related to how the networking service operates. The core_plugin specifies that the modular layer 2 plugin is going to be used. This plugin enables OpenStack to perform advanced layer 2 networking operations but requires a mechanism for driving these operations (a layer 2 device/agent, i.e., a switch). OpenStack supports several “mechanism drivers” or L2 agents. In this deployment you will opt for linuxbridge, a Linux kernel module that provides switching functionality in software (a software switch is often referred to as a bridge). You will also be including an l3 agent to enable layer 3 communications, so you have specified the router service plugin. Your team may use this in future to create a virtual router and configure a DHCP service that will provide automatic IP addressing to your private cloud VM instances.

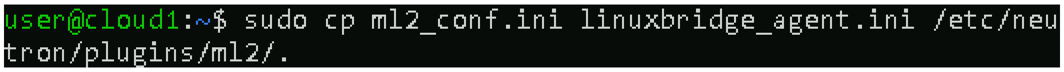
3. At the command prompt**, type** **`sudo cp neutron.conf /etc/neutron/.`** and **press Enter**.



```
user@cloud1:~$ sudo cp neutron.conf /etc/neutron/.
```

Command to install the neutron configuration file

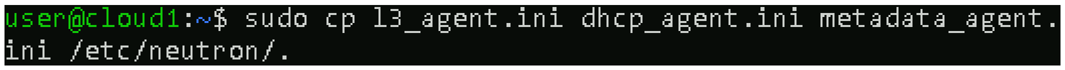
4. At the command prompt, **type** **`sudo cp ml2_conf.ini linuxbridge_agent.ini /etc/neutron/plugins/ml2/.`** and **press Enter** to configure layer 2 network connectivity.



```
user@cloud1:~$ sudo cp ml2_conf.ini linuxbridge_agent.ini /etc/neu
tron/plugins/ml2/.
```

Command to install some networking configuration files

5. At the command prompt, **type** **`sudo cp l3_agent.ini dhcp_agent.ini metadata_agent.ini /etc/neutron/.`** and **press Enter**.



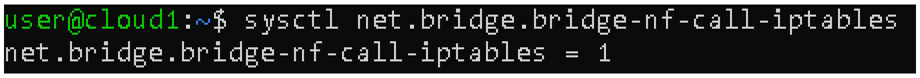
```
user@cloud1:~$ sudo cp l3_agent.ini dhcp_agent.ini metadata_agent.
ini /etc/neutron/.
```

Command to install more networking configuration files

Note: With these commands, all the updated configuration files have been moved into place. For the networking to work correctly, the Linux kernel needs to have bridge filters enabled. This provides the Linux node with the capability to filter and classify packets/frames, necessary for it to perform its switching function. This enablement can be accomplished with one command and then verified with a couple of others.

6. At the command prompt, **type** `sudo modprobe br_netfilter` and **press Enter** to enable br_netfilter.

7. At the command prompt, **type** `sysctl net.bridge.bridge-nf-call-iptables` and **press Enter**.



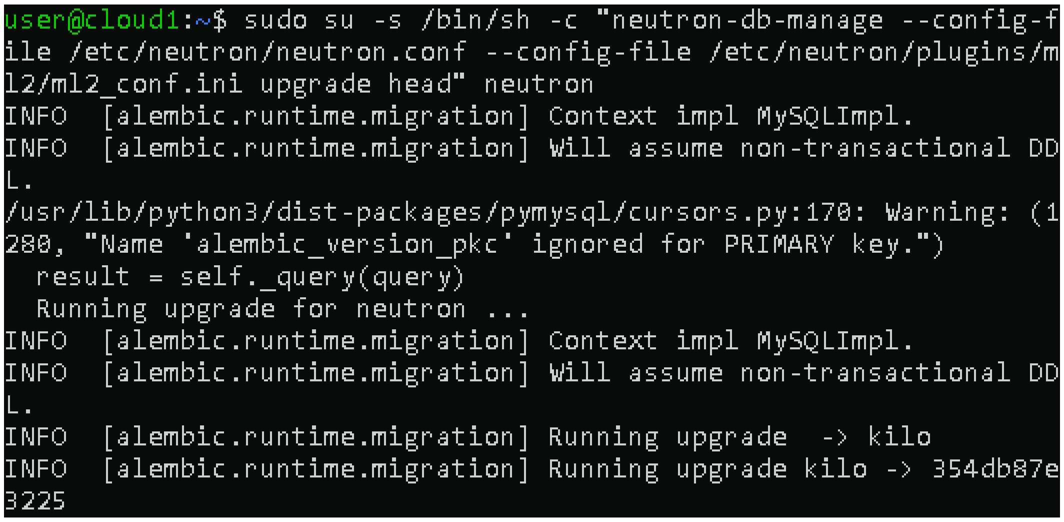
```
user@cloud1:~$ sysctl net.bridge.bridge-nf-call-iptables
net.bridge.bridge-nf-call-iptables = 1
```

Command to verify the kernel module is enabled

8. At the command prompt, **type** `sysctl net.bridge.bridge-nf-call-ip6tables` and **press Enter**.

Note: The first command loads the kernel module if needed. The last two commands show the current state of the kernel for the named properties. They should both return 1, indicating that the netfilter is enabled for both IPv4 and IPv6 traffic.

9. At the command prompt, **type** `sudo su -s /bin/sh -c "neutron-db-manage --config-file /etc/neutron/neutron.conf --config-file /etc/neutron/plugins/ml2/ml2_conf.ini upgrade head" neutron` and **press Enter**.



```
user@cloud1:~$ sudo su -s /bin/sh -c "neutron-db-manage --config-f
ile /etc/neutron/neutron.conf --config-file /etc/neutron/plugins/m
l2/ml2_conf.ini upgrade head" neutron
INFO  [alembic.runtime.migration] Context impl MySQLImpl.
INFO  [alembic.runtime.migration] Will assume non-transactional DD
L.
/usr/lib/python3/dist-packages/pymysql/cursors.py:170: Warning: (1
280, "Name 'alembic_version_pkc' ignored for PRIMARY key.")
  result = self._query(query)
  Running upgrade for neutron ...
INFO  [alembic.runtime.migration] Context impl MySQLImpl.
INFO  [alembic.runtime.migration] Will assume non-transactional DD
L.
INFO  [alembic.runtime.migration] Running upgrade  -> kilo
INFO  [alembic.runtime.migration] Running upgrade kilo -> 354db87e
3225
```

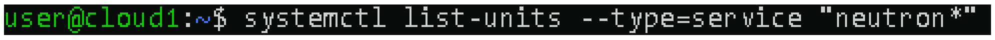
Command to initialize the neutron database

Note: Now that you have completed the neutron configuration, it is time to update your stack and put your new configuration online. While this is a more complex-looking command than those used for the other services, its goal is simply to populate the neutron database with your configuration changes and initialize it to a working state.
Since the nova and neutron services are closely tied together, you will restart nova and enable neutron. You will then verify that the neutron service is running using the systemctl command.

10. At the command prompt, **type** `sudo systemctl restart nova-api` and **press Enter** to restart the nova-api service.

11. At the command prompt, **type** `sudo systemctl enable --now neutron-server` and **press Enter** to enable and start the neutron server.

12. At the command prompt, **type** `systemctl list-units --type=service "neutron*"` and **press Enter**.



```
user@cloud1:~$ systemctl list-units --type=service "neutron*"
```

Command to verify the neutron service is active

Note: The list should show a single neutron-server service that is loaded, active, and running. Now to verify your nova service is also in good shape.

13. At the command prompt, **type** `openstack compute service list --service nova-compute` and **press Enter**.

Note: The output from this command should show a single nova-compute node at cloud2.

14. **Make a screen capture** showing that cloud2 is an active compute node for the stack.

Note: Compute nodes can be added once the stack is set up. Additional nodes can be discovered using the command

sudo su -s /bin/sh -c "nova-manage cell_v2 discover_hosts --verbose" nova

Finally, you have completed setup of the OpenStack system. Now you can to create virtual machines and networks and begin fleshing out your private cloud environment. However, you will first configure a service that will make this enterprise much easier. OpenStack’s horizon service provides a web interface that can be used to manage instances in your private cloud.

15. At the command prompt, **type** `sudo vim /etc/openstack-dashboard/local_settings.py` and **press Enter** to edit the settings file for the dashboard web application.

16. In the editor, **change** the **value of CACHES** as shown below.

```
CACHES = {
      'default': {
            'BACKEND':
'django.core.cache.backends.memcached.MemcachedCache',
            'LOCATION': 'cloud1:11211',
       }
   }
```

Note: The only significant change here is to update the location to point at the running Memcached server.

17. In the editor, **uncomment and change the SESSION_ENGINE** line to match the one below.
    `SESSION_ENGINE = 'django.contrib.sessions.backends.cache'`

18. In the editor, **change** the **OPENSTACK_HOST** value to equal `"cloud1"`.

19. In the editor, **edit the OPENSTACK_KEYSTONE_URL** line and **add four lines to match the following**:

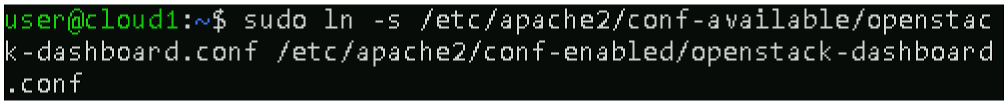
```
OPENSTACK_KEYSTONE_URL = "http://%s:5000/v3" % OPENSTACK_HOST
OPENSTACK_KEYSTONE_MULTIDOMAIN_SUPPORT = True
OPENSTACK_KEYSTONE_DEFAULT_DOMAIN = "Default"
OPENSTACK_KEYSTONE_DEFAULT_ROLE = "user"
OPENSTACK_API_VERSIONS = { "identity": 3, "image": 2, "volume": 3 }
```

Note: Here you have specified some details about your Keystone configuration, the most important of which is the host and endpoint URL for your keystone configuration. Horizon needs to know how to reach the Keystone service, otherwise it will not be able to authenticate users (read: cannot login!). The %s indicates a substitution for OPENSTACK_HOST, which will resolve to localhost (this controller node, cloud1). Following that, you provided the port your keystone service is running on (:5000) and the API endpoint it is reachable at (/v3).

20. In the editor, **type** `:wq` and **press Enter** to save and close the changes.

Note: The horizon service is also proxied by the Apache web server. Therefore, you can use a command similar to the others to enable it.

21. At the command prompt, **type** **`sudo ln -s /etc/apache2/conf-available/openstack-dashboard.conf /etc/apache2/conf-enabled/openstack-dashboard.conf`** and **press Enter** to link the configuration into the enabled area.



```
user@cloud1:~$ sudo ln -s /etc/apache2/conf-available/openstac
k-dashboard.conf /etc/apache2/conf-enabled/openstack-dashboard
.conf
```

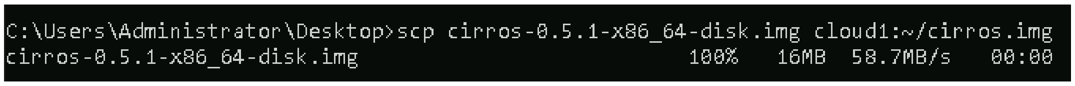
Command to enable the dashboard service

22. At the command prompt, type **`sudo systemctl reload apache2`** and **press Enter** to restart the web server.

Note: To create instances of virtual machines, you need to have some images available. A simple test image that can be uploaded to the glance service has been placed on vWorkstation. The image is in the file cirros-0.5.1-x86_64-disk.img on the Desktop. This is a small Linux operating system image that is designed to be lightweight and useful for testing.

23. On the vWorkstation desktop, **double-click** the **Command Prompt shortcut** to open a new command prompt window.

24. At the new command prompt, **type** **`scp cirros-0.5.1-x86_64-disk.img cloud1:~/cirros.img`** and **press Enter** to copy the image to cloud1.



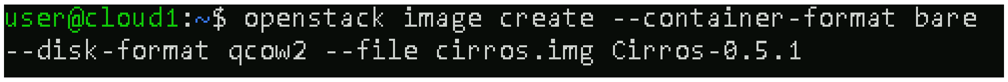
```
C:\Users\Administrator\Desktop>scp cirros-0.5.1-x86_64-disk.img cloud1:~/cirros.img
cirros-0.5.1-x86_64-disk.img                                   100%   16MB  58.7MB/s   00:00
```

Command to copy the cirros image

25. **Close** the **new command prompt window**.

Note: Now the image needs to be added to the OpenStack deployment.

26. At the command prompt for cloud1, **type** `openstack image create --container-format bare --disk-format qcow2 --file cirros.img Cirros-0.5.1` and **press Enter** to create the image.

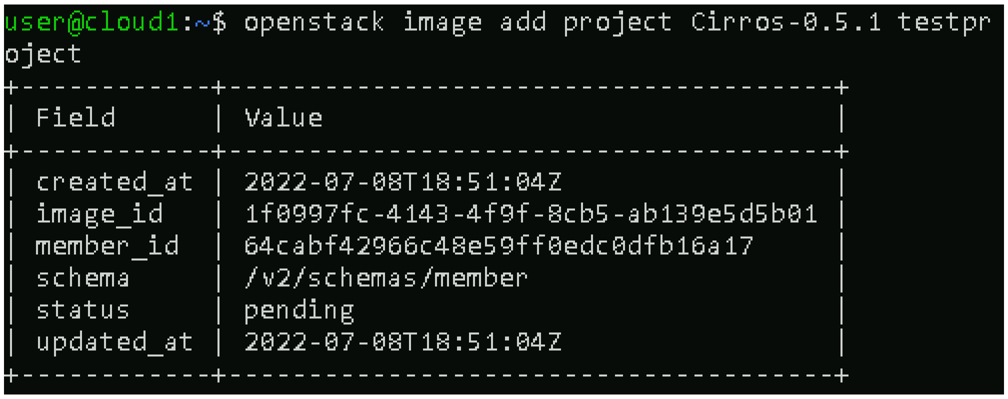


```
user@cloud1:~$ openstack image create --container-format bare
--disk-format qcow2 --file cirros.img Cirros-0.5.1
```

Command to add the Cirros image to OpenStack

Note: This command registers the image with the glance service and makes it available. However, it does not make it available for creating instances yet. All instances are owned by projects. For an image to be available, it needs to be added to the project.

27. At the command prompt, **type** `openstack image add project Cirros-0.5.1 testproject` and **press Enter** to make the image available in the test project.

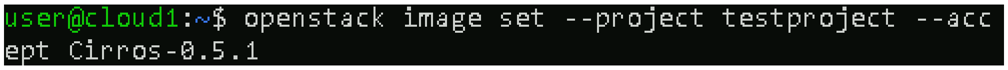


```
user@cloud1:~$ openstack image add project Cirros-0.5.1 testpr
oject
+------------+--------------------------------------+
| Field      | Value                                |
+------------+--------------------------------------+
| created_at | 2022-07-08T18:51:04Z                 |
| image_id   | 1f0997fc-4143-4f9f-8cb5-ab139e5d5b01 |
| member_id  | 64cabf42966c48e59ff0edc0dfb16a17     |
| schema     | /v2/schemas/member                   |
| status     | pending                              |
| updated_at | 2022-07-08T18:51:04Z                 |
+------------+--------------------------------------+
```

Command to add the Cirros image to the test project

Note: The output of the last command should show the image as pending for the project. Thus, it is still not ready. Someone who has the authorization needs to accept the image into the project. This can be accomplished with the image set command.

28. At the command prompt, **type** `openstack image set --project testproject --accept Cirros-0.5.1` and **press Enter** to accept the image for the test project.



```
user@cloud1:~$ openstack image set --project testproject --acc
ept Cirros-0.5.1
```

Command to accept the Cirros image into the test project

Note: Now you have finally gotten the image to be available for the test project. You now have enough set up that it is time to actually look at the system through the web interface.

29. On vWorkstation, **double-click** the **Firefox icon** to open a web browser.

Firefox icon

30. **Navigate** the web browser to **http://172.30.10/horizon**.

31. **Make a screen capture** of the horizon login page.

32. On the web page, **log in** using **default** for the domain, **testuser** for the username, and **testpass** for the password.

Note: After all of this you should get a successful login. If you navigate around, there are many options for starting to define instances and networks. However, there are a few more things to do before instances can actually be created. You will explore this more in the Challenge and Analysis section, if assigned.

Challenge and Analysis

Note: The following scenario is provided to allow independent, unguided work – similar to what you will encounter in a real-world situation.

Part 1: Set Up a Flavor for Nova Compute Instances

Your team's research has determined that you need to set up something called a flavor as a step toward creating instances in your OpenStack deployment. Based on your teammate's notes, a flavor is a configuration for the resources available to a virtual machine instance, controlling things like the ram and number of CPUs.

Research the OpenStack guides online and determine the command line needed to create a new flavor that uses very few resources (say 64 MB of ram, 1 GB of hard disk, and 1 CPU) and is suitable for testing the Cirros image.

On the cloud1 machine, use the command you discovered above to create your new instance flavor; then, use the command **`openstack flavor list`** to output its details to the console.

Make a screen capture showing your new flavor in the flavor list output.

Part 2: Add an Additional Nova Compute Resource

Your team leader would also like to have a sense of what is involved with scaling the system to have more compute resources available. The simplest way to do this is to add more compute nodes to cell 1. You are tasked with configuring cloud3 (172.30.0.30) as a second compute node.

Using the OpenStack guides and the steps that were used on cloud2, configure cloud3 as an additional computing node in cell 1.

Once you are finished, on cloud1 **execute** **`openstack compute service list --service nova-compute`** to list all available nova compute nodes.

Make a screen capture showing both cloud2 and cloud3 in the list of available computing nodes.