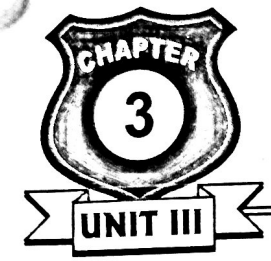

OpenStack

University Prescribed Syllabus

Introduction to OpenStack, OpenStack test-drive, Basic OpenStack operations, OpenStack CLI and APIs, Tenant model operations, Quotas, Private cloud building blocks, Controller deployment, Networking deployment, Block Storage deployment, Compute deployment, deploying and utilizing OpenStack in production environments, Building a production environment, Application orchestration using OpenStack Heat.

3.1 Introduction to OpenStack

What is OpenStack ?

- OpenStack is a set of software tools for building and managing cloud computing platforms for public and private clouds. Backed by few of the biggest companies in software development and hosting, as well as thousands of individual community members, many thinks that OpenStack is the future of cloud computing. OpenStack is managed by the OpenStack Foundation, a non-profit that oversees for both development and community-building around the project.
- The following Fig. 3.1 shows several of the resource components that OpenStack coordinates to create public and private clouds. As the figure, OpenStack doesn't replace these resource providers; it simply manages them, through control points built into the framework.
- With an OpenStack cloud you can
  - Harness the resources of physical and virtual servers, networks, and storage systems
  - Efficiently manage clouds of resources through tenants, quotas, and user roles
  - Provide a common interface to control resources regardless of the underlying
  - vendor subsystem

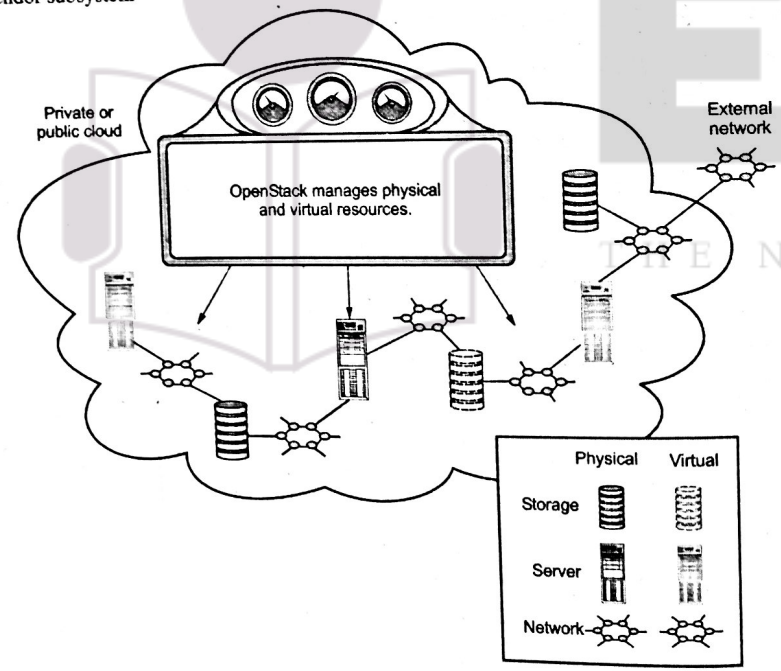

Fig. 3.1.1 : Open Stack is a cloud Operating System

- At first glance, OpenStack doesn't look like a traditional operating system, but then
- A second time, the "cloud" does not look like a normal computer. One must take a step back and consider the fundamental benefits of an operating system.
- Before there were operating systems or even hardware-level abstraction languages (Assembly), programs were written in the language (binary machine code) of a specific computer.
- Then traditional operating systems came along and, among other things, allowed users to standardize not just application code, but also the management functions of the hardware.
- Administrators could now manage hardware instances using a common interface, developers could write code for a common system, and users only had to learn a single user interface.
- This held true regardless of underlying hardware, as long as the operating system remained the same.
- In the evolution of computers, the development and proliferation of operating systems gave rise to the field of systems engineering and administration.

3.1.1 Understanding Cloud Computing and OpenStack

- The cloud is all about providing computing for end users in a remote environment, where the actual software runs as a service on reliable and scalable servers rather than on each end-user's computer.
- Cloud computing can refer to a lot of different things, but typically the industry talks about running different items "as a service" software, platforms, and infrastructure.
- OpenStack falls into the latter category and is considered Infrastructure as a Service (IaaS).
- In contrast, public cloud IaaS resources are owned and operated by third-party service providers, like Amazon AWS, Microsoft Azure, and the like.
- Providing infrastructure means that OpenStack makes it easy for users to quickly add new instance, upon which other cloud components can run. Typically, the infrastructure then runs a "platform" upon which a developer can create software applications that are delivered to the end users.

3.1.2 OpenStack Components

GQ. 3.1.1 What are the key components of OpenStack ?

UQ. 3.1.2 What are the key components of OpenStack ? (MU - April 19, 5 Marks)

- There are many more projects in various stages of development, but these are the foundational components of OpenStack.
- OpenStack is made up of many different moving parts. Hence of its open nature, anyone can add additional components to OpenStack to help it to meet their needs.
- But the OpenStack community has collaboratively identified nine key components that are a part of the "core" of OpenStack, which are distributed as a part of any OpenStack system and officially maintained by the OpenStack community.

(a) Nova is the primary computing engine behind OpenStack. Nova is used for deploying and managing large numbers of virtual machines and other instances to handle computing tasks.

(b) **Swift** is a storage system for objects and files. Rather than the traditional idea of a referring to files by their location on a disk drive, developers can instead refer to a unique identifier referring to the file or piece of information and let OpenStack decide where to store this information. This makes scaling easy, as developers don't have the worry about the capacity on a single system behind the software. It also allows the system, rather than the developer, to worry about how best to make sure that data is backed up in case of the failure of a machine or network connection.

(c) **Cinder** is a block storage component, that is more analogous to the traditional notion of a computer being able to access specific locations on a disk drive. This is traditional way of accessing files might be important in scenarios in which data access speed is the most important consideration.

(d) **Neutron** provides the networking capability for OpenStack. It helps to ensure that every of the components of an OpenStack deployment can communicate with one another quickly and efficiently.

(e) **Horizon** is the dashboard behind OpenStack. It is the only graphical interface to OpenStack, hence for users wanting to give OpenStack a try, which might be the 1st component they actually "see." Developers can access each of the components of OpenStack individually through an application programming interface (API), even though the dashboard provides system administrators a look at what is going on in the cloud, and to manage it as needed.

(f) **Keystone** provides identity services for OpenStack. It is essentially a central list of each of the users of the OpenStack cloud, mapped against each of the services provided by the cloud, that they have permission to use. It provides multiple means of access, meaning developers can easily map their existing user access methods against Keystone.

(g) **Glance** provides image services to OpenStack. In this case, "images" states to images (or virtual copies) of hard disks. Glance allows that images to be used as templates when deploying new virtual machine instances.

(h) **Ceilometer** provides telemetry services, which allow the cloud to provide billing services to individual users of the cloud. It also keeps a verifiable count of each user's system usage of each of the all components of an OpenStack cloud. Think metering and usage reporting.

(i) **Heat** is the orchestration component of OpenStack, which allows developers to store the requirements of a cloud application in a file that defines what resources are necessary for that application. In this way, it helps to manage the infrastructure required for a cloud service to run.

3.1.3 Benefits of OpenStack using Cloud

GQ. 3.1.3 Explain the benefits of using OpenStack Cloud.

- Any enterprise that wants to flourish and increase the efficiency of their operations needs to look for inventive ways to do it. Fortunately, with this digital age, it is possible to achieve agility, flexibility & efficiency in several ways.
- OpenStack is by far one of the best ways to accomplish all those things at the same time. It stands out among the plethora of cloud options, as OpenStack provides the flexibility of technology virtualization like no other solutions available in the market.

- Hundreds of companies rely on OpenStack to run their businesses smoothly every day while reducing costs and overheads. In addition to this, OpenStack is the leading open source option for building public cloud environments. It does not matter whether your company is a multibillion-dollar publicly traded enterprise or a small startup, you can use OpenStack public clouds with services that compete with major public cloud providers.

Great industry support

- Back in 2010, OpenStack was formed by NASA as a platform for allowing organizations to provide free and open source cloud computing services running on standard commodity hardware.
- Since its inception, giants in IT industry such as IBM, Intel, Red Hat, AMD etc have pledged their support and provided investment for it to develop. Hence, OpenStack has a thriving community of developers that collaborate with the users to design the platform best suited for any business needs.

Compatibility

- OpenStack's APIs are designed to be compatible with public cloud platforms such as Amazon Web Services.
- For plenty of businesses, this provides the option of porting IaaS client applications from AWS to OpenStack-based IaaS providers with minimal effort.

Scalability

- Scalability is best defined as the property of a system to continue to work as it increases in size and workload demands.
- In case of OpenStack public clouds, it offers a significant degree of scalability for enterprises OpenStack itself is a highly scalable system. It can enable enterprises to spin up and spin down servers on-demand.

Security

- Securing an OpenStack cloud is not unlike securing any other IT infrastructure. It requires a high set of tools and skills and a good understanding of security.
- In OpenStack, high level of security can be achieved through role-based access controls. Access and resource utilization can be controlled at the level of users, roles, and projects.

Easy to manage panel

- OpenStack has a control panel that provides visibility, control and easy access to power management tools.
- Thus making it very easy for the users to monitor and manage their cloud services and allowing users and administrators to have a clear overview for the management of resource usage, and currently active VM instances.

3.2 OpenStack Test-drive

- In test-drive OpenStack through the use of DevStack, a rapid OpenStack deployment tool, and answers these questions.

- DevStack lets you interact with OpenStack on a small scale that's representative of a larger deployment. You can quickly deploy or "stack" (as fellow OpenStackers call it) components and evaluate them for production use cases.
- DevStack helps you deploy the same OpenStack components found in large multiserver environments on a single server, as shown in following figure. Without knowing a great deal about OpenStack and without the need for a bunch of hardware, you can use DevStack to get the OpenStack experience, just on a smaller scale.

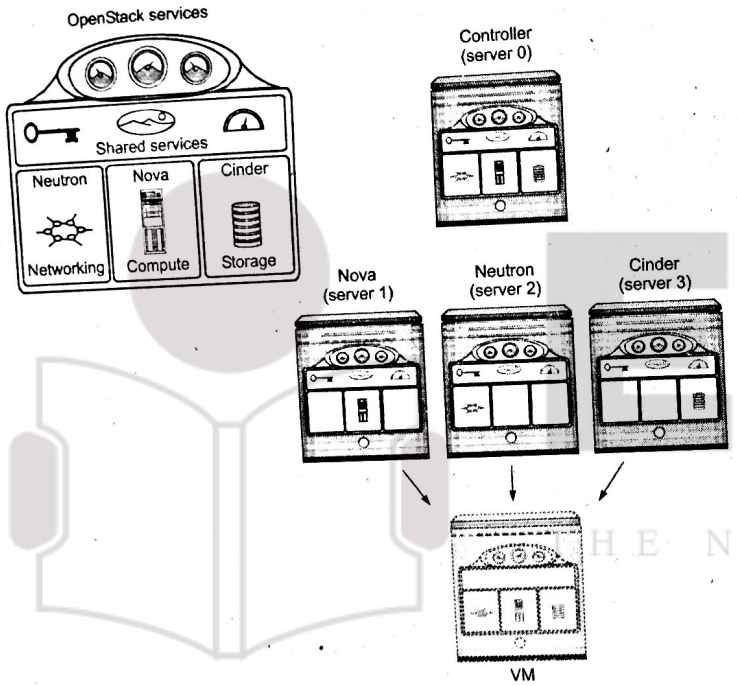

Fig. 3.2.1 : Multiserver OpenStack

- The figure shows several components, including Cinder, Nova, and Neutron,deployed on an arbitrary number of nodes. OpenStack uses codenames for its components; therefore the codename Cinder refers to storage components, Nova to compute components, and Neutron to network components.
- What you need to know now is that OpenStack is creating of several core components that can be distributed among nodes (servers) based on the intended design.

3.2.1 What is DevStack ?

UQ. 3.2.1 What is DevStack ? Explain the steps to install DevStack. (MU - April 19, 5 Marks)

- DevStack was made to make the job of deploying OpenStack in test and development environments quicker, easier, and more understandable, but the ease with which it allows users to deploy OpenStack build it a natural starting point for learning the framework.
- DevStack is a collection of documented Bash (command-line interpreter) shell scripts that are used to prepare an environment for, configure, and deploy OpenStack.
- The choice of using a shell-scripting language for DevStack was deliberate.
- The code is intended to be read by humans and computers alike, and it's used as a source of documentation by developers.
- Developers of OpenStack components can document dependencies outside of raw code segments, and users can understand how these dependencies must be provided in a working system.

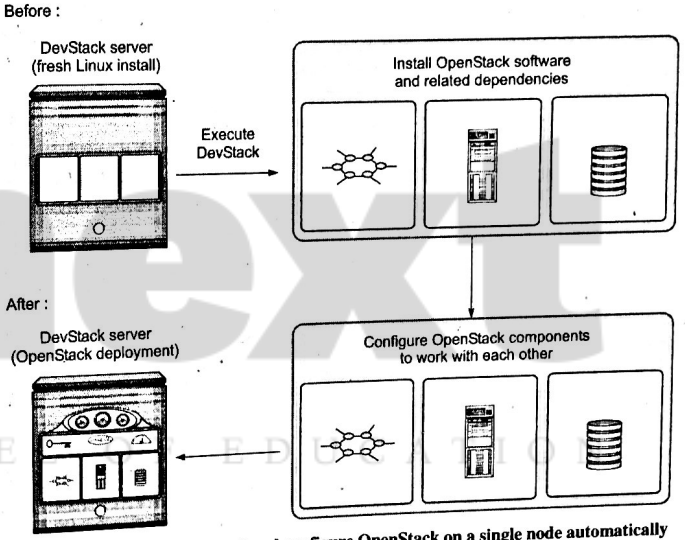

Fig. 3.2.2 : DevStack will install and configure OpenStack on a single node automatically

- Despite the daunting size and complexity of the OpenStack framework, DevStack makes things look easy.
- In the Fig. 3.2.2 may look like an over simplification, but it's an accurate example of the function of DevStack.
- Users with very little experience with virtualization, storage, networking, or frankly Linux, can quickly get a single-server OpenStack environment working. In many ways, DevStack does for OpenStack what OpenStack can do for infrastructure; it simplifies and abstracts it.
- You'll prepare an environment and deploy OpenStack using DevStack. You don't need to know much about Linux, storage, or networking to deploy a working single-server OpenStack environment.

- Using this deployment, we'll walk through ways you can interact with OpenStack, and this will give you some familiarity with both the components and the overall system.
- Then we'll discuss the OpenStack tenant model, which is how OpenStack logically separates, controls, and assigns resources to projects. In OpenStack terminology, tenant and project can be used interchangeably.
- Finally, you'll take what you've learned and create a virtual machine in a virtualization environment.

3.2.3 The Steps to Install DevStack

- DevStack is a series of extensible scripts used to quickly bring up a complete OpenStack environment based on the latest versions of everything from git master.
- It is used interactively as a development environment and as the basis for lot of the OpenStack project's functional testing.

(a) Install one of the supported Linux Distributions.

(b) Add Stack User (optional)

DevStack should be run as a non-root user with sudo enabled (standard logins to cloud images such as "ubuntu" or "cloud-user" are usually fine).

If you are not using a cloud image, you can create a separate stack user to run DevStack with

```
$sudouseradd -s /bin/bash -d /opt/stack -m stack
```

Since this user will be making many changes to your system, it should have sudo privileges:

```
$echo"stack ALL=(ALL) NOPASSWD: ALL"|sudo tee /etc/sudoers.d/stack
$sudosu - stack
```

(a) Download DevStack

```
$ git clone https://opendev.org/openstack/devstack
$cd devstack
```

The devStack repo contains a script that installs OpenStack and templates for configuration files.

(b) Create a local.conf

Create a local.conf file with four passwords preset at the root of the devStackgit repo.

```
[[local|localrc]]
ADMIN_PASSWORD=secret
DATABASE_PASSWORD=$ADMIN_PASSWORD
RABBIT_PASSWORD=$ADMIN_PASSWORD
SERVICE_PASSWORD=$ADMIN_PASSWORD
```

This is the minimum required config to get started with DevStack.

(c) Start the install

```
$ ./stack.sh
```


This will take around 15 - 20 minutes, largely depending on the speed of your internet connection. Many git trees and packages will be installed during this process.

3.2.3 Using the OpenStack Dashboard

- Three primary ways to interface with OpenStack:
  1. OpenStack Dashboard - A web-based GUI, introduced in this section
  2. OpenStack CLI - Component-specific command-line interfaces,
  3. OpenStack APIs - RESTful (web) services
- Regardless of interface method, all interactions will make their way back to the OpenStack APIs.
- For most people, their first hands-on exposure to OpenStack will be through the Dashboard. In fact, the majority of end users will use the Dashboard exclusively, so that's the access method we'll discuss here. System administrators and programmers will need to understand how to access the CLI and APIs.

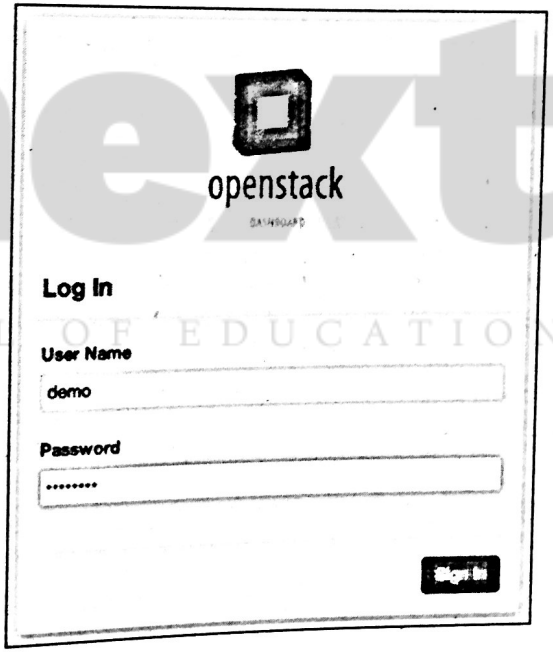

Fig. 3.2.3 : Dashboard login screen

- Go ahead and access the Dashboard by entering the following URL into your browser: http://<your host ip> You should be presented with the login screen shown in Fig. 3.2.3, where you can enter the following user name and password:

- User name: demo
- Password: devstack

- The demo user simulates an unprivileged user. If you don't see the login screen, it's likely that an error occurred during the stacking process.

3.3 Basic OpenStack Operations

GQ. 3.3.1 List and explain the basic OpenStack operations tasks.

- In this we are focuses on operational exercises, so examples are based on the OpenStack command-line interface (CLI).
- If you have systems administration experience, you'll certainly appreciate the ability to script a repetitive function, such as creating a thousand users. The OpenStack APIs can also be used for those tasks, and they will be briefly introduced. As you'll discover, if you can perform an operation with the CLI, you can easily perform the same operation with an API directly.
- For the examples, we'll stick with using the CLI; this is constructed so that you can walk through the examples using either an API or the Dashboard once you understand the concepts demonstrated through the CLI.
- The CLI also has the added benefit of using separate applications for each OpenStack component. While at first this might seem like a bad thing, it will help you better understand which component is responsible for what.
- These and other objects were created by DevStack, but tenants, users, networks, and other objects won't be created for you automatically in manual deployment. Here we'll walk through the process of creating the necessary objects to take a test drive in a tenant you create.

3.4 OpenStack CLI and APIs

3.4.1 Using the OpenStack CLI

GQ. 3.4.1 Explain the OpenStack Command Line Interface (CLI).

- As, you can run CLI commands, you must first set the appropriate environment variables in your shell. Environment variables tell the CLI how and where to recognize you.
- You can provide input for those variables directly to the CLI, but for the sake of clarity, all examples will be shown with the appropriate environment variables in place. To set these variables, run the commands shown in the next listing in your shell. Each time you log in to a session, you'll have to set your environment variables.
- To set these variables, run the commands shown in the next listing in your shell. Each time you log in to a session, you'll have to set your environment variables.

Listing 3.1 Set environmental variables

```
source /opt/stack/python-novaclient/tools/nova.bash_completion
source openrc demo demo
```

Sets variables for shell completion so that pressing tab after entering "something /bo" completes "something /boot"

Run this command from ~/devstack directory. When you run OpenStack CLI commands, your identity will be (user) <demo> in (tenant) <demo>.

- To make sure your variables have been properly set, you should test if you can run an OpenStack CLI command. In your shell, run the nova image-list command, as shown in listing 3.2. This CLI command reads the environment variables you just set and uses them as identification. If you're properly identified and have rights to do so, the CLI will query OpenStack Compute (Nova) for your currently available image-list.

Listing 3.2 Setting variables and executing a first CLI command

```
devstack@devstack:~/devstack$ source \
  /opt/stack/python-novaclient/tools/nova.bash_completion
devstack@devstack:~/devstack$ source openrc demo demo
devstack@devstack:~/devstack$ nova image-list
+----+---------------------------------------+--------+--------+
| ID | Name                                  | Status | Server |
+----+---------------------------------------+--------+--------+
| 4. | Ubuntu 12.04                          | ACTIVE |        |
| f. | cirros-0.3.1-x86_64-uec               | ACTIVE |        |
| a. | cirros-0.3.1-x86_64-uec-kernel        | ACTIVE |        |
| a. | cirros-0.3.1-x86_64-uec-ramdisk       | ACTIVE |        |
+----+---------------------------------------+--------+--------+
```

This simple command will list your Nova images.

- You should now be able to run OpenStack CLI commands as the demo user in the demo tenant.
- Using the command in listing 3.3, you can create a new OpenStack instance, just like you did in the Dashboard.

Listing 3.3 Launching an instance from the CLI

```
nova boot \
  --flavor <flavor_id> \
  --image <image_id> \
  <instance name>
```

Tells Nova that you want to boot/create an instance

Specifies <flavor_id> (size) as shown by the command nova flavor-list

Specifies <image_id> as shown by the command nova image-list

Specifies name of your instance

- When you run this command, you'll get results something like the following:

```
nova boot \
--flavor 3 \
--image 48ab76e9-c3f2-4963-8e9b-6b22a0e9c0cf \
Test_Instance_3
+------------------------------------+--------+--------+
+------------------------------------+----------------+
| Property                           | Value          |
+------------------------------------+----------------+
| OS-DCF:diskConfig                  | MANUAL         |
| OS-EXT-AZ:availability_zone        | nova           |
| OS-EXT-STS:power_state             | 0              |
| OS-EXT-STS:task_state              | scheduling     |
| OS-EXT-STS:vm_state                | building       |
| OS-SRV-USG:launched_at             | -              |
| OS-SRV-USG:terminated_at           | -              |
| accessIPv4                         |                |
| accessIPv6                         |                |
...                                  .
...                                  .
+------------------------------------+----------------+
```

- You can do everything with the OpenStack CLI that you can using the Dashboard, and more. In the preceding example, you performed the Nova boot command, which provisioned a new VM.
- To get help with more-advanced Nova commands, use the following command: nova help COMMAND (replacing COMMAND with the command you're interested in). There are similar command-line utilities for Keystone, Glance, Neutron, and so on.

3.4.2 Using the OpenStack APIs

- The component-specific APIs interface with a number of sources, including other APIs and relational databases. This also holds true for the Dashboard.
- All OpenStack interactions eventually lead back to the OpenStack API layer.
- It could certainly be argued that the inherent vendor neutrality provided by the
- OpenStack APIs is OpenStack's greatest benefit. People who integrate external systems or debug OpenStack code will find themselves looking at the API layer.
- The thing to keep in mind is that all roads lead to the OpenStack APIs. If you have further interest in them, see the sidebar, "Debug CLI/Expose API."

- To get started using the OpenStack APIs directly, you can follow the example in listing 3.4. This command will query the OpenStack APIs for information, which will be returned in JavaScript Object Notation (JSON) format. Python is used to parse the JSON so it can be read on your screen.

Listing 3.4 Executing a first API command

```
curl -s -X POST http://10.0.2.32:5000/v2.0/tokens \
 -d '{"auth": {"passwordCredentials": \
{"username":"demo", "password":"devstack"}, \
"tenantName":"demo"}}' -H "Content-type: application/json" | \
python -m json.tool
```

Substitute your IP for 10.0.2.32.

Debug CLI/Expose API

Every CLI command will output its API command if the debug flag is set. To enable debugging for a specific CLI command, pass the --debug argument before any other variables as shown here:

```
devstack@devstack:~$ nova --debug image-list

REQ: curl -i 'http://10.0.2.32:5000/v2.0/tokens'
-X POST -H "Content-Type: application/json"
-H "Accept: application/json"
-H "User-Agent: python-novaclient"
-d '{"auth": {"tenantName": "admin", "passwordCredentials":
{"username": "admin", "password": "devstack"}}}'

...
```

- Now that you understand the mechanics of the OpenStack CLI and APIs, you're ready to put these skills to use. In the next section we'll walk through creating a new tenant (project) using the CLI.
- This is an operational function you'll perform for each new department, user, or project you want to separate from a more general tenant.

3.5 Tenant model Operations

GQ. 3.5.1 Explain Tenant network with suitable diagram.

What Is Tenant

A tenant is a group of users who are sharing a common access with specific privileges to the software instance.

Tenant model operations

- Let's take an example; A person can't be a resident of a hotel unless they have a room, so you can think of tenants as hotel rooms.
- As an alternative to beds and a TV, Hotel OpenStack provides computational resources.

- A hotel room (as tenant) which is configurable (like single or double beds, a suite or a room, and so on).
- The number of resources (vCPU, RAM, storage), images (tenant-specific software images), and the configuration of the network are all based on tenant-specific configurations.
- Users are independent of tenants, but users may hold roles for specific tenants. A single user might be holding the role of administrator in multiple tenants. Every time a new user is added to OpenStack, they must be assigned a tenant.
- Every time a new instance (VM) is created, it must be created in a tenant. Management of all OpenStack resources is based on the management of tenant resources.

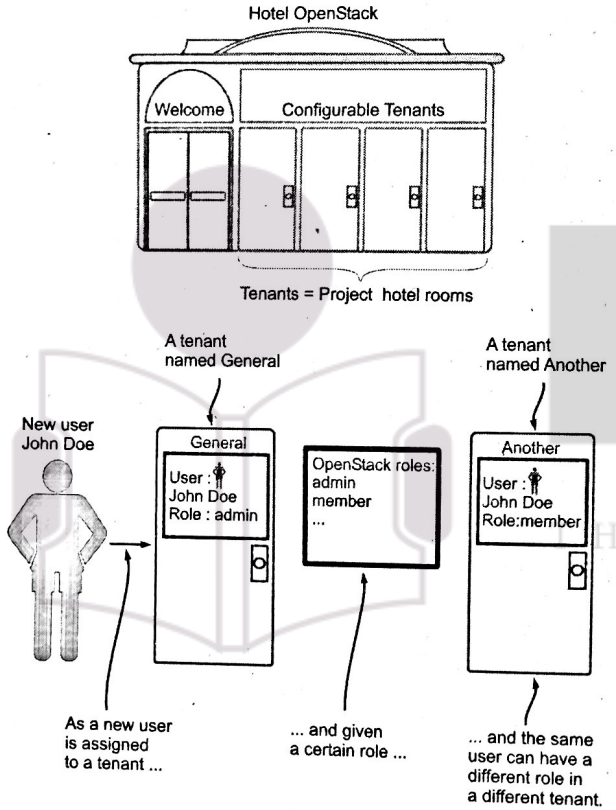

Fig. 3.5.1 : The relation of tenants, users, and roles in OpenStack Identity (Keystone)

- Hence your access to OpenStack resources is based on tenant configurations; you must understand how to create new tenants, users, roles, and quotas.

- In the next few subsections, you'll walk through creating a new tenant and all the related objects that go with it, from scratch. As an OpenStack administrator, this will be a common task. A department or even a project might be a new tenant.
- Tenants will be the fundamental way that you divide and manage configurations and resources in OpenStack.

3.5.1 Tenant Networks

- First, though, you need to understand the basic differences between how traditional "flat" networks are configured for virtual and physical machines and how OpenStack networking will be demonstrated.
- The term flat refers to the absence of a virtual routing tier as part of the virtual server platform; in traditional configurations, the VM has direct access to a network, as if you plugged a physical device into a physical network switch.
- In this type of deployment, all network services (Dynamic Host Configuration Protocol (DHCP), load balancing, routing, and so on) beyond simply switching (Open System Interconnection (OSI) Model, Layer 2), must be provided outside of the virtual environment.
- For most systems administrators, this type of configuration will be very familiar, but this is not how we'll demonstrate the power of OpenStack. You can make OpenStack Networking behave like a traditional flat network, but that approach will limit the benefits of the OpenStack framework.
- In this section, you'll build an OpenStack tenant network from scratch Fig. 3.5.2 is example of the differences between a more traditional network and the type of network you'll build.

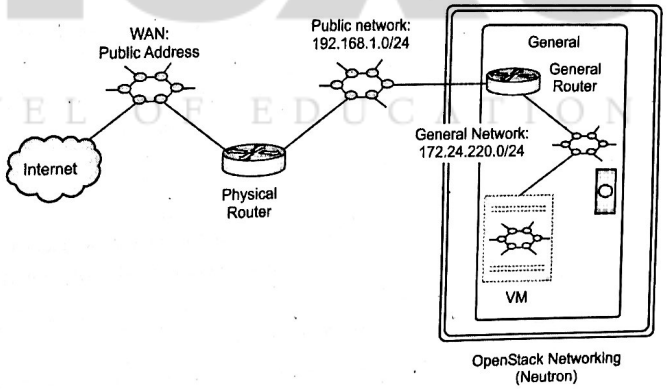

Fig. 3.5.2 : OpenStack tenant network

- Note that compared to the traditional flat network, the OpenStack tenant network includes an additional router that resides within the virtual environment.

- The addition of the virtual router in the tenant separates the internal network, shown as GENERAL_NETWORK, from the external network, shown as PUBLIC_NETWORK. VMs communicate with each other using the internal network, and the virtual router, shown as GENERAL_ROUTER, uses the external network for communication outside the tenant.

3.6 Quota

GQ. 3.6.1 *Explain Quotas in OpenStack.*

UQ. 3.6.2 *Explain the concept of Quota in OpenStack.* (MU - April 19, 5 Marks)

What is Quota ?

- To prevent system capacities from being exhausted without notification, you can set up quotas. Quotas are operational limits.
- For example, the number of gigabytes allowed per tenant can be controlled to ensure that a single tenant cannot consume all the disk space. Quotas are currently enforced at the tenant (or project) level, rather than the user level.
- Quotas are applied on the tenant and tenant-user level to limit the amount of resources any one tenant can use.
- When you create a new tenant, a default quota is applied. Likewise, when you add users to your tenant, the tenant quota is applied to them.
- By default, all users have the same quota as the tenant quota, but you have the option of reducing a user's quota in a tenant independently of the overall tenant quota.
- Consider the case where you have an application administrator and database administrator sharing the same tenant for a project.
- You might want to assign half of the tenant resource to each user. On the other hand, if you increase a user's quota in a tenant in excess of the tenant quota, the tenant quota will increase to match the new user value.

Tenant quotas

- To modify quota settings, you'll need to know the tenant ID you want to work with and a user ID that's currently in that tenant. The following example shows how you can list all tenants on the system and find the tenant ID:

  devstack@devstack:~/devstack$ keystone tenant-list

- Using the command-line interface (CLI), you can manage quotas for the OpenStack Compute service and the Block Storage service.
- Typically, default values are changed because a tenant requires more than the OpenStack default of 10 volumes per tenant, or more than the OpenStack default of 1 TB of disk space on a compute node.

Note

To view all tenants, run:

```
$ openstack project list
+----------------------------------+----------+
| ID                               | Name     |
+----------------------------------+----------+
| a981642d22c94e159a4a6540f70f9f8 | admin    |
| 934b662357674c7b9f5e4ec6ded4d0e | tenant01 |
| 7bc1dbfd7d284ec4a856ea1eb82dca8 | tenant02 |
| 9c554aaef7804ba49e1b21cbd97d218 | services |
+----------------------------------+----------+
```

Showing the Compute quota for a tenant

```
nova quota-show \
  --tenant 9932bc0607014caeab4c3d2b94d5a40c
```

Tells OpenStack Compute (Nova) to show quota information

Specifies tenant ID for quota query

The quota information will be displayed as follows.

```
+-----------------------------+-------+
| Quota                       | Limit |
+-----------------------------+-------+
| instances                   | 10    |
| cores                       | 10    |
| ram                         | 51200 |
| floating_ips                | 10    |
| fixed_ips                   | -1    |
| metadata_items              | 128   |
| injected_files              | 5     |
| injected_file_content_bytes | 10240 |
| injected_file_path_bytes    | 255   |
| key_pairs                   | 100   |
| security_groups             | 10    |
| security_group_rules        | 20    |
+-----------------------------+-------+
```

Updating the Compute quota for tenant

```
nova quota-update \
 --<quota_key> <quota_value> \
 <tenant-id>
```

Tells Compute that you want to update quota values

Assigns new quota value to a specific quota item

Specifies tenant whose quota you want to update

- A list of quota keys can be obtained by displaying the quota values, as shown previously in listing 3.21. In the following example, the cores<quota_key> is updated:

```
devstack@devstack:~/devstack$ nova quota-update \
 --cores 20 \
 9932bc0607014caeab4c3d2b94d5a40c
```

Sets <quota-key> to cores and <quota-value> to 20

Specifies ID for General tenant

- You've now successfully updated your tenant quota, and your users can now start assigning additional resources. If you rerun the command shown in listing 3.21, you'll see that the quota has been updated.

☞ Tenant-user quotas

- In some cases, there might be only a single user in a tenant. In these cases, you'd only need quota management on the tenant level. But what if there are multiple users in one tenant? OpenStack provides the ability to manage quotas for individual users on a tenant level. This means that an individual user can have separate quotas for each tenant of which they are a member.
- Suppose that one of your users with a role on a specific tenant is only responsible for a Single instance. Despite being responsible for only one instance, this user has on several occasions added additional instances to this tenant.
- The additional instances count against the overall tenant quota, so although the user in question should only have one instance, they might have several. To prevent this from happening, you can adjust the user's quota for the tenant, and not the entire tenant quota. The following listing displays the existing quota for a particular user

☞ Show Tenant User

```
nova quota-show \
 --user <user_id> \
 --tenant <tenant_id>
```

Specifies user ID in a tenant for query

Specifies tenant ID for query

- The following example shows the user ID related to johndoe, which you created in the Sub section "Creating a user," and the tenant ID related to the General tenant, which you created in the subsection "Creating a tenant." Your actual IDs will differ from the examples listed here.


```
devstack@devstack:~/devstack$ nova quota-show \
 --user 21b27d5f7ba04817894d290b660f3f44 \
 --tenant 9932bc0607014caeab4c3d2b94d5a40c
+-----------------------------+-------+
| Quota                       | Limit |
+-----------------------------+-------+
| instances                   | 10    |
| cores                       | 10    |
| ram                         | 51200 |
| floating_ips                | 10    |
| fixed_ips                   | -1    |
| metadata_items              | 128   |
| injected_files              | 5     |
| injected_file_content_bytes | 10240 |
| injected_file_path_bytes    | 255   |
| key_pairs                   | 100   |
| security_groups             | 10    |
| security_group_rules        | 20    |
+-----------------------------+-------+
```

Specifies tenant ID for General

Specifies user ID for johndoe

- As you can see, the user quotas are the same size as the original tenant quota. By default, users added to a tenant can use all resources assigned to that tenant.
- For this tenant, you updated the cores value in a previous example, but that only updated the tenant quota, which, as you can see, doesn't automatically increase a user's tenant quota.
- Assume that user johndoe is a problem user that you want to restrict to running a single instance in the General tenant. The next listing shows the command you can use to do this.

```
nova quota-update \
 --user <user_id> \
 --<quota_key> <quota_value> \
 <tenant_id>
```

Specifies user's ID

Assigns a new quota value to a quota item for a specific user

Specifies tenant ID

- In the following example, the user johndoe is configured an instance-quota-metric limitation of 1 instance (instance - quota = 1) in the General tenant:

```
devstack@devstack:~/devstack$ nova quota-update \
 --user 21b27d5f7ba04817894d290b660f3f44 \
 --instances 1 \
 9932bc0607014caeab4c3d2b94d5a40c
```

Specifies user-id related to johndoe user

Specifies tenant-id related to General tenant

Sets <quota-key> to "instances" and <quota-value> to 1 instance

- Now You have restricted the user johndoe to running a single instance. You might further want to restrict the resources this user can utilize for their individual instance, such as limiting the number of cores to 4.

☞ Quota in Nova

- Nova uses a quota system for setting limits on resources such as number of instances or amount of CPU that a specific project or user can use.
- Quotas are enforced by making a claim, or reservation, on resources when a request is made, such as creating a new server. If the claim fails, the request is rejected.
- If the reservation succeeds, then the operation progresses until such a point that the reservation is either converted into usage (the operation was successful) or rolled back (the operation failed).
- Typically, the quota reservation is made in the nova-api service and the usage or rollback is performed in the nova-compute service, at least when dealing with a server creation or move operation.
- Quota limits and usage can be retrieved via the limits REST API.

☞ Checking quota

- When calculating limits for a given resource and tenant, the following checks are made in order:
- Depending on the resource, is there a tenant-specific limit on the resource in either the quotas or project_user_quotas tables in the database. If so, use that as the limit. You can create these resources by doing:

```
openstackquotaset--instances5<project>
```

- Check to see if there is a hard limit for the given resource in the quota classes table in the database for the default quota class. If so, use that as the limit. You can modify the default quota limit for a resource by doing :

```
openstack quota set --class --instances 5 default
```

- If the above does not provide a resource limit, then rely on the quota_* configuration options for the default limit.

➲ 3.6.1 Architecture of Neutron

UQ. 3.6.3 Explain architecture of Neutron. **(MU - April 19, 5 Marks)**

- The architecture of Neutron is simple, but it is with the agents and plugins where the real magic happens! Neutron architecture has been presented in the following diagram :

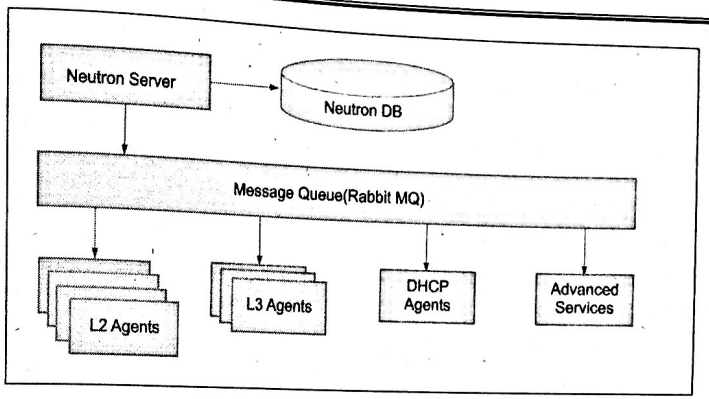

Fig . 3.6.1 : Architecture of Neutron

- Let's discuss the role of the different components in a little detail.

☞ The Neutron server

The function of this component is to be the face of the entire Neutron environment to the outside world. It essentially is made up of three modules:

- **REST service :** The REST service accepts API requests from the other components and exposes all the internal working details in terms of networks, subnets, ports, and so on. It is a WSGI application written in Python and uses port 9696 for communication.
- **RPC service :** The RPC service communicates with the messaging bus and its function is to enable a bidirectional agent communication.
- **Plugin :** A plugin is best described as a collection of Python modules that implement a standard interface, which accepts and receives some standard API calls and then connects with devices downstream. They can be simple plugins or can implement drivers for multiple classes of devices.
- The plugins are further divided into core plugins, which implement the core Neutron API, which is Layer 2 networking (switching) and IP address management. If any plugin provides additional network services, we call it the service plugin -- for example, Load Balancing as a Service (LBaaS), Firewall as a Service (FWaaS), and so on.
- As an example, Modular Layer 2 (ML2) is a plugin framework that implements drivers and can perform the same function across ML2 networking technologies commonly used in data centers. We will use ML2 in our installation to work with Open vSwitch (OVS).

☞ L2 agent

The L2 agent runs on the hypervisor (compute nodes), and its function is simply to wire new devices, which means it provides connections to new servers in appropriate network segments and also provides notifications when a device is attached or removed. In our install, we will use the OVS agent.

L3 agent

The L3 agents run on the network node and are responsible for static routing, IP forwarding, and other L3 features, such as DHCP.

Understanding the basic Neutron process

Let's take a quick look at what happens when a new VM is booted with Neutron. This shows all the steps that take place during the Layer 2 stage:

1. Boot VM start.
2. Create a port and notify the DHCP of the new port.
3. Create a new device (virtualization library – libvirt).
4. Wire port (connect the VM to the new port).
5. Complete boot.

3.6.2 Nova Architecture

UQ. 3.6.4 Explain architecture of Nova. (MU - April 19, 8 Marks)

- Nova is architected as a distributed application with many components, but the majority of these are custom-written Python daemons of two varieties:
  - Web Server Gateway Interface (WSGI) applications to receive and mediate API calls
  - Worker daemons to carry out orchestration tasks
- However, there are two essential pieces of the architecture that are neither custom written nor Python-based: the messaging queue and the database.
- These two components facilitate the asynchronous orchestration of complex tasks through message passing and information sharing. Piecing this all together we get a picture like Fig. 3.6.2.

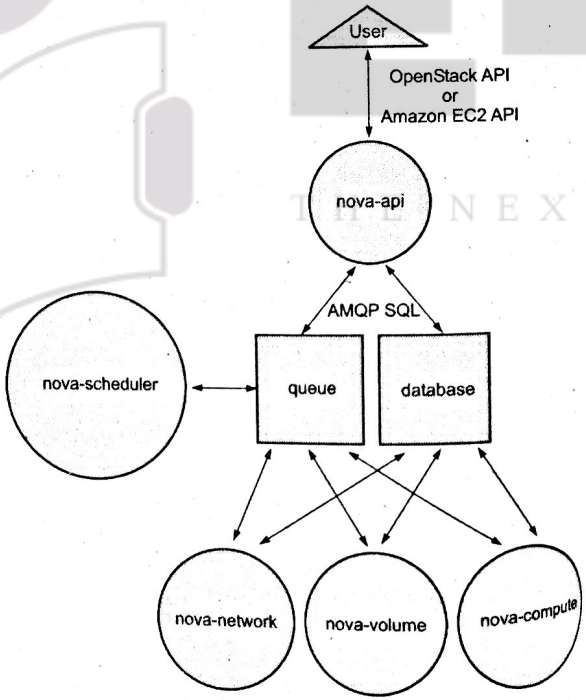

Fig. 3.6.2 : Architecture of Nova

- This complicated diagram can be summed up in three sentences:
  - End users who want to use Nova to create compute instances call *nova-api* with OpenStack API or EC2 API requests.
  - Nova daemons exchange information through the queue (actions) and database (information) to carry out these API requests.
  - Glance is a completely separate service that Nova interfaces through the Glance API to provide virtual disk imaging services.
- Now that we've seen the overview of the processes and their interactions, let's take a closer look at each component.

API

- The nova-api daemon is the heart of Nova. You may see it illustrated on many pictures of Nova as API and "Cloud Controller."
- While this is partly true, cloud controller is really just a class (specifically the Cloud Controller in nova/api/ec2/cloud.py) within the nova-api daemon. Its primary purpose is to accept and fulfill incoming API requests.

Scheduler

- The nova-scheduler process is conceptually the simplest piece of code in Nova: it takes a virtual machine instance request from the queue and determines where it should run (specifically, which compute server host it should run on).
- In practice, however, this will grow to be the most complex piece, as it needs to factor in the current state of the entire cloud infrastructure and apply complicated algorithms to ensure efficient usage.
- To that end, nova-scheduler implements a pluggable architecture that lets you choose (or write) your own algorithm for scheduling.

Compute Worker

- The nova-compute process is primarily a worker daemon that creates and terminates virtual machine instances.
- The process by which it does so is fairly complex, but the basics are simple: accept actions from the queue and then perform one or a series of virtual machine API calls to carry them out while updating state in the database.
- An example of this would be nova-compute accepting a message from the queue to create a new instance and then using the libvirt library to start a new KVM instance.

Volume Worker

- As you can gather by the name, nova-volume manages the creation, attaching, and detaching of persistent volumes to compute instances (similar in functionality to Amazon's Elastic Block Storage).
- It can use volumes from a variety of providers such as iSCSI or AoE.

Network Worker

- The nova-network worker daemon is very similar to nova-compute and nova-volume. It accepts networking tasks from the queue and then performs system commands to manipulate the network (such as setting up bridging interfaces or changing iptables rules).
- Nova defines two different types of IP addresses for an instance: Fixed IPs and Floating IPs. These can be broadly thought of as private IPs (fixed) and public IPs (floating).
- Fixed IPs are assigned on instance startup and remain the same during their entire lifetimes. Floating IPs are dynamically allocated and associated to a domain to allow outside connectivity.

Queue

- The queue provides a central hub for passing messages between daemons. This is currently implemented with RabbitMQ today, but theoretically could be any AMPQ message queue supported by the Python ampqlib and carrot libraries.
- Nova creates several types of message queues to facilitate communication between the various daemons. These include: topics queues, fanout queues, and host queues. Topics queues allow messages to be broadcast to the number of particular class of worker daemons.

Database

The database stores most of the configuration and run-time state for a cloud infrastructure. This includes the instance types that are available for use, instances in use, networks available, and projects.

3.7 Private Cloud Building Blocks

GQ. 3.7.1 Explain Private cloud building blocks.

- The three main building blocks which required to build a private cloud are
  - Physical Layer
  - Virtualization Layer
  - Cloud Management Layer
- The physical layer-is the foundation layer of the cloud reference model. The process of building a cloud infrastructure is typically initiated with the cloud service provider setting up the physical hardware resources of the cloud infrastructure.
- The physical layer comprises compute, storage, and network resources, which are the fundamental physical computing resources that make up a cloud infrastructure.
- Physical compute systems host the applications that a provider offers as services to consumers and also execute the software used by the provider to manage the cloud infrastructure and deliver services. A cloud provider also offers compute systems to consumers for hosting their applications in the cloud.
- Storage systems store business data and the data generated or processed by the applications deployed on the compute systems.

- Storage capacity may be offered along with a compute system or separately (for example, in case of cloud-based backup). Networks connect compute systems with each other and with storage systems.
- A network, such as a local area network (LAN), connects physical compute systems to each other, which enables the applications running on the compute systems to exchange information.
- A storage network connects compute systems to storage systems, which enables the applications to access data from the storage systems. If a cloud provider uses physical computing resources from multiple cloud data centers to provide services, networks connect the distributed computing resources enabling the data centers to work as a single large data center.
- Networks also connect multiple clouds to one another as in case of the hybrid cloud model - to enable them to share cloud resources and services. Based on several requirements such as performance, scalability, cost, and so on, a cloud provider has to make a number of decisions while building the physical layer, including choosing suitable compute, storage, and network products and components, and the architecture and design of each system.
- The subsequent lessons describe various physical components and architectures that are available to cloud providers to build the physical layer.
- A compute system is a computing platform (hardware, firmware, and software) that runs platform and application software. Examples of physical compute systems include desktops, laptops, servers, mobile devices, and so on.
- A compute system consists of processor (s), memory, I/O devices, and a collection of software to perform computing operations. The software includes the operating system (OS), file system, logical volume manager, device drivers, and so on.
- The OS may include the other software or they can be installed individually. The OS manages the physical components and application execution, and provides a user interface (UI) for users to operate and use the compute system.
- A compute system typically comprises the following key physical hardware components assembled inside an enclosure:
- **Processor :** A processor, also known as a Central Processing Unit (CPU), is an integrated circuit (IC) that executes the instructions of a software program by performing fundamental arithmetical, logical, and input/output operations. A common processor/instruction set architecture is the x86 architecture with 32-bit and 64-bit processing capabilities. Modern processors have multiple cores (independent processing units), each capable of functioning as an individual processor.
- **Random-Access Memory (RAM) :** The RAM or main memory is a volatile data storage device internal to a compute system. The RAM holds the software programs for execution and the data used by the processor.
- **Read-Only Memory (ROM) :** A ROM is a type of semiconductor memory that contains the boot firmware (that enables a compute system to start), power management firmware, and other device-specific firmware.
- Motherboard : A motherboard is a printed circuit board (PCB) to which all compute system components connect. It has sockets to hold components such as the microprocessor chip, RAM, and ROM. It also has network ports, I/O ports to connect devices such as keyboard, mouse, and printers, and essential circuitry to carry out computing operations.

- A motherboard may additionally have integrated components, such as a graphics processing unit (GPU), a network interface card (NIC), and adapters to connect to external storage devices.
- **Chipset :** A chipset is a collection of microchips on a motherboard and it is designed to perform specific functions. The two key chipset types are Northbridge and Southbridge. Northbridge manages processor access to the RAM and the GPU, while Southbridge connects the processor to different peripheral ports, such as USB ports.
- On a compute system, a cloud provider deploys software such as the self-service portal, the application software and platform software that are offered as services (PaaS and SaaS) to consumers, virtualization software, cloud infrastructure management software, and so on. The provider also enables consumers to deploy their platform software and business applications on the compute systems.
- Providers typically install compute virtualization software (hypervisor) on a computer system and create multiple virtual compute systems, known as virtual machines (VMs), each capable of running its own OS. In this case, the hypervisor performs compute system management tasks and allocates the compute system's resources, such as processor and memory, dynamically to each VM. The provider allocates the VMs running on a hypervisor to consumers for deploying their applications. The provider may pre install an OS on a VM or may enable the consumers to install an OS of their choice.
- Data created by individuals, businesses, and applications need to be persistently stored so that it can be retrieved when required for processing or analysis. A storage system is the repository for saving and retrieving electronic data and is integral to any cloud infrastructure.
- A storage system has devices, called storage devices (or storage) that enable the persistent storage and the retrieval of data. Storage capacity is typically offered to consumers along with compute systems.
- Apart from providing storage along with compute systems, a provider may also offer storage capacity as a service (Storage as a Service), which enables consumers to store their data on the provider's storage systems in the cloud. This enables the consumers to leverage cloud storage resources for purposes such as data backup and long-term data retention.
- A cloud storage infrastructure is typically created by logically aggregating and pooling the storage resources from one or more data centers to provide virtual storage resources. Cloud storage provides massive scalability and rapid elasticity of storage resources. The cloud storage infrastructure is typically shared by multiple tenants or consumers which improves the utilization of storage resources.

☞ Virtual Private Cloud

- A Virtual Private Cloud is implemented using a shared data center infrastructure of hardware and software. The data center is most likely off-premises. It is shared with multiple organizations. If the data center is not shared, that is a Private Cloud.
- The upper layers of the Cloud Computing Stack (PaaS and SaaS) in a Virtual Private Cloud is dedicated to the organization. The lower IaaS is shared in a Virtual Private Cloud.
- A Virtual Private Cloud may participate in a Hybrid Cloud.
- The following figure uses Customer Relationship Management (CRM) Software as a Service (SaaS) to illustrate a Virtual Private Cloud.

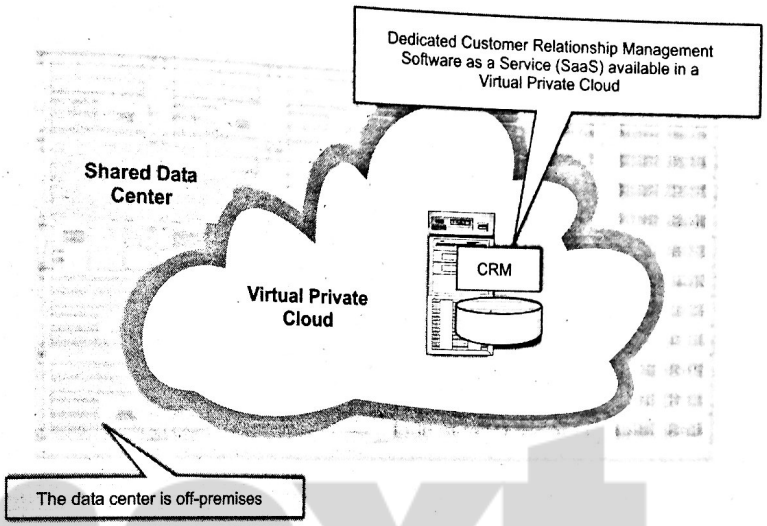

Fig. 3.7.1 : Uses Customer Relationship Management (CRM) Software as a Service (SaaS)

☞ Private cloud management tools

- For private cloud management, organizations often use in-house tools. Such tools can include platform-specific management software, such as VMTurbo Operations Manager and Embotics vCommander.
- Some private cloud management tools offer sophisticated software frameworks to manage complex private and hybrid cloud deployments. These tools include Microsoft System Center Virtual Machine Manager (VMM) for Hyper-V, VMware vCloud Suite, Red Hat CloudForms and Citrix CloudPlatform.

3.8 Controller Deployment

GQ. 3.8.1 *Explain Controller deployment in OpenStack.*

UQ. 3.8.2 *Explain various deployment models of OpenStack.* (MU - April 19, 5 Marks)

In this section describes how to install and configure the Compute service, code-named nova, on the controller node.

Nodes

The deployment examples refer one or more of the following nodes:

- Controller : Contains control plane components of OpenStack services and their dependencies.
  - Two network interfaces: management and provider.
  - Operational SQL server with databases necessary for each OpenStack service.

- Operational message queue service.
- Operational OpenStack Identity (keystone) service.
- Operational OpenStack Image Service (glance).
- Operational management components of the OpenStack Compute (nova) service with appropriate configuration to use the Networking service.
- OpenStack Networking (neutron) server service and ML2 plug-in.

– Network: Contains the OpenStack Networking service layer-3 (routing) component. High availability options may include additional components.

- Three network interfaces: management, overlay, and provider.
- Openstack Networking layer-2 (switching) agent, layer-3 agent, and any dependencies.

– Compute : Contains the hypervisor component of the OpenStack Compute service and the OpenStack Networking layer-2, DHCP, and metadata components. High-availability options may include additional components.

- Two network interfaces: management and provider.
- Operational hypervisor components of the OpenStack Compute (nova) service with appropriate configuration to use the Networking service.
- OpenStack Networking layer-2 agent, DHCP agent, metadata agent, and any dependencies.

– Each building block defines the quantity and types of nodes including the components on each node.

3.8.1 Prerequisites

Before you install and configure the Compute service, you must create databases, service credentials, and API endpoints.

1. To create the databases, complete these steps:
   – Use the database access client to connect to the database server as the root user

```
$ mysql -u root -p
```

   – Create the nova_api and nova databases:

```
CREATE DATABASE nova_api;
CREATE DATABASE nova;
```

   – Grant proper access to the databases:

```
GRANT ALL PRIVILEGES ON nova_api.* TO 'nova'@'localhost' \
  IDENTIFIED BY 'NOVA_DBPASS';
GRANT ALL PRIVILEGES ON nova_api.* TO 'nova'@'%' \
  IDENTIFIED BY 'NOVA_DBPASS';
GRANT ALL PRIVILEGES ON nova.* TO 'nova'@'localhost' \
```



```
  IDENTIFIED BY 'NOVA_DBPASS';
GRANT ALL PRIVILEGES ON nova.* TO 'nova'@'%' \
  IDENTIFIED BY 'NOVA_DBPASS';
```

   – Replace NOVA_DBPASS with a suitable password.
   – Exit the database access client.

2. Source the admin credentials to gain access to admin-only CLI commands:

```
$ . admin-openrc
```

3. To create the service credentials, complete these steps:
   – Create the nova user:

```
$ openstack user create --domain default \
  --password-prompt nova
User Password:
Repeat User Password:
+-----------+----------------------------------+
| Field     | Value                            |
+-----------+----------------------------------+
| domain_id | e0353a670a9e496da891347c589539e9 |
| enabled   | True                             |
| id        | 8c46e4760902464b889293a74a0c90a8 |
| name      | nova                             |
+-----------+----------------------------------+
```

   – Add the admin role to the nova user:

```
$ openstack role add --project service --user nova admin
```

Note : This command provides no output.

   – Create the nova service entity:

```
$ openstack service create --name nova \
  --description "OpenStack Compute" compute
+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description | OpenStack Compute                |
| enabled     | True                             |
| id          | 060d59eac51b4594815603d75a00aba2 |
```

```
| name      | nova                             |
| type      | compute                          |
+-----------+----------------------------------+
```

4. Create the Compute service API endpoints:

```
$ openstack endpoint create --region RegionOne \
  compute public http://controller:8774/v2.1/%\(tenant_id\)s
+--------------+----------------------------------------+
| Field        | Value                                  |
+--------------+----------------------------------------+
| enabled      | True                                   |
| id           | 3c1caa473bfe4390a11e7177894bcc7b       |
| interface    | public                                 |
| region       | RegionOne                              |
| region_id    | RegionOne                              |
| service_id   | e702f6f497ed42e6a8ae3ba2e5871c78       |
| service_name | nova                                   |
| service_type | compute                                |
| url          | http://controller:8774/v2.1/%(tenant_id)s |
+--------------+----------------------------------------+

$ openstack endpoint create --region RegionOne \
  compute internal http://controller:8774/v2.1/%\(tenant_id\)s
+--------------+----------------------------------------+
| Field        | Value                                  |
+--------------+----------------------------------------+
| enabled      | True                                   |
| id           | e3c918de680746a586eac1f2d9bc10ab       |
| interface    | internal                               |
| region       | RegionOne                              |
| region_id    | RegionOne                              |
| service_id   | e702f6f497ed42e6a8ae3ba2e5871c78       |
| service_name | nova                                   |
| service_type | compute                                |
| url          | http://controller:8774/v2.1/%(tenant_id)s |
```



```
+--------------+----------------------------------------+

$ openstack endpoint create --region RegionOne \
  compute admin http://controller:8774/v2.1/%\(tenant_id\)s
+--------------+----------------------------------------+
| Field        | Value                                  |
+--------------+----------------------------------------+
| enabled      | True                                   |
| id           | 38f7af91666a47cfb97b4dc790b94424       |
| interface    | admin                                  |
| region       | RegionOne                              |
| region_id    | RegionOne                              |
| service_id   | e702f6f497ed42e6a8ae3ba2e5871c78       |
| service_name | nova                                   |
| service_type | compute                                |
| url          | http://controller:8774/v2.1/%(tenant_id)s |
+--------------+----------------------------------------+
```

3.8.2 Install and Configure Components

Note : Default configuration files vary by distribution. You might need to add these sections and options rather than modifying existing sections and options. Also, an ellipsis (...) in the configuration snippets indicates potential default configuration options that you should retain.

1. Install the packages :

```
# apt-get install nova-api nova-conductor nova-consoleauth \
  nova-novncproxy nova-scheduler
```

2. Edit the /etc/nova/nova.conf file and complete the following actions:
   - In the [DEFAULT] section, enable only the compute and metadata APIs:

```
[DEFAULT]
...
enabled_apis = osapi_compute,metadata
```

   - In the [api_database] and [database] sections, configure database access:

```
[api_database]
...
connection = mysql+pymysql://nova:NOVA_DBPASS@controller/nova_api
```

```
[database]
...
connection = mysql+pymysql://nova:NOVA_DBPASS@controller/nova
```

- Replace NOVA_DBPASS with the password you chose for the Compute databases.
- In the [DEFAULT] and [oslo_messaging_rabbit] sections, configure RabbitMQ message queue access:

```
[DEFAULT]
...
rpc_backend = rabbit

[oslo_messaging_rabbit]
...
rabbit_host = controller
rabbit_userid = openstack
rabbit_password = RABBIT_PASS
```

- In the [DEFAULT] and [keystone_authtoken] sections, configure Identity service access:

```
[DEFAULT]
...
auth_strategy = keystone

[keystone_authtoken]
...
auth_uri = http://controller:5000
auth_url = http://controller:35357
memcached_servers = controller:11211
auth_type = password
project_domain_name = default
user_domain_name = default
project_name = service
username = nova
password = NOVA_PASS
```

Replace NOVA_PASS with the password you chose for the nova user in the Identity service.

Note : Comment out or remove any other options in the [keystone_authtoken] section.

- In the [DEFAULT] section, configure the my_ip option to use the management interface IP address of the controller node :

```
[DEFAULT]
...
my_ip = 10.0.0.11
```

- In the [DEFAULT] section, enable support for the Networking service:

```
[DEFAULT]
...
use_neutron = True
firewall_driver = nova.virt.firewall.NoopFirewallDriver
```

Note : By default, Compute uses an internal firewall driver. Since the Networking service includes a firewall driver, you must disable the Compute firewall driver by using the nova.virt.firewall.NoopFirewallDriver firewall driver.

- In the [vnc] section, configure the VNC proxy to use the management interface IP address of the controller node :

```
[vnc]
...
vncserver_listen = $my_ip
vncserver_proxyclient_address = $my_ip
```

- In the [glance] section, configure the location of the Image service API:

```
[glance]
...
api_servers = http://controller:9292
```

- In the [oslo_concurrency] section, configure the lock path:

```
[oslo_concurrency]
...
lock_path = /var/lib/nova/tmp
```

- Due to a packaging bug, remove the logdir option from the [DEFAULT] section.

3. Populate the Compute databases:

```
# su -s /bin/sh -c "nova-manage api_db sync" nova
# su -s /bin/sh -c "nova-manage db sync" nova
```

Note : Ignore any deprecation messages in this output.

3.8.3 Finalize installation

Restart the Compute services :

```
# service nova-api restart
# service nova-consoleauth restart
# service nova-scheduler restart
# service nova-conductor restart
# service nova-novncproxy restart
```

3.9 Networking Deployment

GQ. 3.9.1 *Explain Networking deployment in OpenStack.*

UQ. 3.9.2 *Explain various deployment models of OpenStack.* (MU - April 19, 5 Marks)

Networks and network interfaces

The deployment examples refer to one or more of the following networks and network interfaces:

- **Management :** Handles API requests from clients and control plane traffic for OpenStack services including their dependencies.
- **Overlay :** Handles self-service networks using an overlay protocol such as VXLAN or GRE.
- **Provider :** Connects virtual and physical networks at layer-2. Typically uses physical network infrastructure for switching/routing traffic to external networks such as the Internet.

Note : For best performance, 10+ Gbps physical network infrastructure should support jumbo frames.

For illustration purposes, the configuration examples typically reference the following IP address ranges:

- Management network: 10.0.0.0/24
- Overlay (tunnel) network: 10.0.1.0/24
- Provider network 1:
  - IPv4: 203.0.113.0/24
  - IPv6: fd00:203:0:113::/64
- Provider network 2:
  - IPv4: 192.0.2.0/24
  - IPv6: fd00:192:0:2::/64
- Self-service networks:
  - IPv4: 192.168.0.0/16 in /24 segments
  - IPv6: fd00:192:168::/48 in /64 segments
- You may change them to work with your particular network infrastructure.

3.10 Block Storage Deployment

GQ. 3.10.1 *Explain Block Storage deployment in OpenStack.*

- Block storage is implemented in OpenStack by the Block Storage service (cinder). Because these volumes are persistent, they can be detached from one instance and re-attached to another instance and the data remains intact. The Block Storage service supports multiple back ends in the form of drivers.

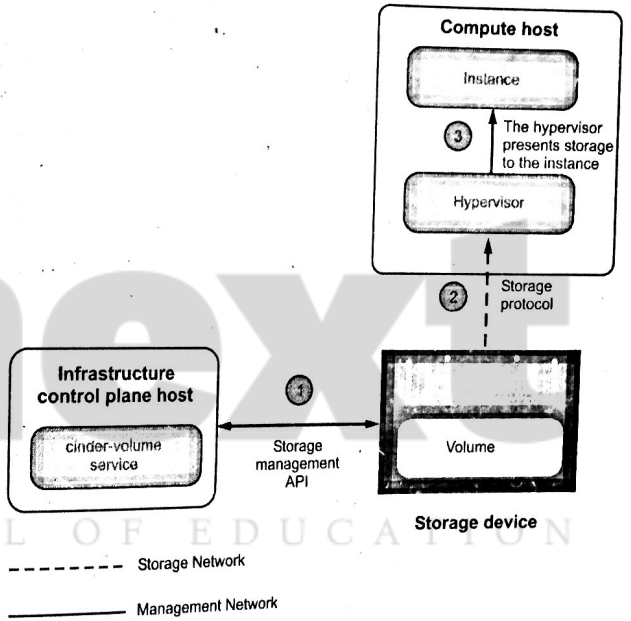

Fig. 3.10.1 : Cinder Storage Overview

- The above diagram shows the following steps.
  1. A volume is created by the assigned cinder-volume service using the appropriate cinder driver. The volume is created by using an API that is presented to the management network.
  2. After the volume is created, the nova-compute service connects the Compute host hypervisor to the volume via the storage network.
  3. After the hypervisor is connected to the volume, it presents the volume as a local hardware device to the instance.
- The OpenStack Block Storage service (Cinder) adds persistent storage to a virtual machine. Block Storage provides an infrastructure for managing volumes, and interacts with OpenStack Compute to provide volumes for instances. The service also enables management of volume snapshots, and volume types.

The Block Storage service consists of the following components :

- **cinder-api :** Accepts API requests, and routes them to the cinder-volume for action.
- **cinder-volume :** Interacts directly with the Block Storage service, and processes such as the cinder-scheduler. It also interacts with these processes through a message queue. The cinder-volume service responds to read and write requests sent to the Block Storage service to maintain state. It can interact with a variety of storage providers through a driver architecture.
- **cinder-scheduler daemon :** Selects the optimal storage provider node on which to create the volume. A similar component to the nova-scheduler.
- **cinder-backup daemon :** The cinder-backup service provides backing up volumes of any type to a backup storage provider. Like the cinder-volume service, it can interact with a variety of storage providers through a driver architecture.
- **Messaging queue :** Routes information between the Block Storage processes.

☞ Install and configure a storage node

Before you install and configure the Block Storage service on the storage node, you must prepare the storage device.

➡ **Note :** Perform these steps on the storage node.

1. Install the supporting utility packages:

```
# apt-get install lvm2
```

➡ **Note :** Some distributions include LVM by default.

2. Create the LVM physical volume /dev/sdb:

```
# pvcreate /dev/sdb
Physical volume "/dev/sdb" successfully created
```

3. Create the LVM volume group cinder-volumes:

```
# vgcreate cinder-volumes /dev/sdb
Volume group "cinder-volumes" successfully created
```

The Block Storage service creates logical volumes in this volume group.

4. Only instances can access Block Storage volumes. However, the underlying operating system manages the devices associated with the volumes. By default, the LVM volume scanning tool scans the /dev directory for block storage devices that contain volumes. If projects use LVM on their volumes, the scanning tool detects these volumes and attempts to cache them which can cause a variety of problems with both the underlying operating system and project volumes. You must reconfigure LVM to scan only the devices that contain the cinder-volume volume group. Edit the /etc/lvm/lvm.conf file and complete the following actions:

- In the devices section, add a filter that accepts the /dev/sdb device and rejects all other devices :

```
devices {
...
filter = [ "a/sdb/", "r/.*/"]
```

- Each item in the filter array begins with a for **accept** or r for **reject** and includes a regular expression for the device name. The array must end with r/.*/ to reject any remaining devices. You can use the **vgs -vvvv** command to test filters.

☞ Warning

If your storage nodes use LVM on the operating system disk, you must also add the associated device to the filter. For example, if the /dev/sda device contains the operating system:

```
filter = [ "a/sda/", "a/sdb/", "r/.*/"]
```

Similarly, if your compute nodes use LVM on the operating system disk, you must also modify the filter in the /etc/lvm/lvm.conf file on those nodes to include only the operating system disk. For example, if the /dev/sda device contains the operating system:

```
filter = [ "a/sda/", "r/.*/"]
```

☞ Install and configure components

1. Install the packages:

```
# apt-get install cinder-volume
```

2. Edit the /etc/cinder/cinder.conf file and complete the following actions:

- In the [database] section, configure database access:

```
[database]
...
connection = mysql+pymysql://cinder:CINDER_DBPASS@controller/cinder
```

Replace CINDER_DBPASS with the password you chose for the Block Storage database.

- In the [DEFAULT] and [oslo_messaging_rabbit] sections, configure RabbitMQ message queue access:

```
...
rpc_backend = rabbit

[oslo_messaging_rabbit]
...
rabbit_host = controller
rabbit_userid = openstack
rabbit_password = RABBIT_PASS
```

Replace RABBIT_PASS with the password you chose for the openstack account in RabbitMQ.

- In the [DEFAULT] and [keystone_authtoken] sections, configure Identity service access:

```
[DEFAULT]
...
auth_strategy = keystone

[keystone_authtoken]
...
auth_uri = http://controller:5000
auth_url = http://controller:35357
memcached_servers = controller:11211
auth_type = password
project_domain_name = default
user_domain_name = default
project_name = service
username = cinder
password = CINDER_PASS
```

Replace CINDER_PASS with the password you chose for the cinder user in the Identity service.

Note : Comment out or remove any other options in the [keystone_authtoken] section.

- In the [DEFAULT] section, configure the my_ip option:

```
[DEFAULT]
...
my_ip = MANAGEMENT_INTERFACE_IP_ADDRESS
```

Replace MANAGEMENT_INTERFACE_IP_ADDRESS with the IP address of the management network interface on your storage node, typically 10.0.0.41 for the first node in the example architecture.

- In the [lvm] section, configure the LVM back end with the LVM driver, cinder-volumes volume group, iSCSI protocol, and appropriate iSCSI service:

```
[lvm]
...
volume_driver = cinder.volume.drivers.lvm.LVMVolumeDriver
volume_group = cinder-volumes
iscsi_protocol = iscsi
iscsi_helper = tgtadm
```


- In the [DEFAULT] section, enable the LVM back end:

```
[DEFAULT]
...
enabled_backends = lvm
```

Note : Back-end names are arbitrary. As an example, this guide uses the name of the driver as the name of the back end.

- In the [DEFAULT] section, configure the location of the Image service API:

```
[DEFAULT]
...
glance_api_servers = http://controller:9292
```

- In the [oslo_concurrency] section, configure the lock path:

```
[oslo_concurrency]
...
lock_path = /var/lib/cinder/tmp
```

Finalize installation

1. Restart the Block Storage volume service including its dependencies:

```
# service tgt restart
# service cinder-volume restart
```

3.11 Compute Deployment

Install and configure a compute node

- This section describes how to install and configure the Compute service on a compute node. The service supports several hypervisors to deploy instances or VMs. For simplicity, this configuration uses the QEMU hypervisor with the KVM extension on compute nodes that support hardware acceleration for virtual machines.
- On legacy hardware, this configuration uses the generic QEMU hypervisor. You can follow these instructions with minor modifications to horizontally scale your environment with additional compute nodes.

Note : This section assumes that you are following the instructions in this guide step-by-step to configure the first compute node. If you want to configure additional compute nodes, prepare them in a similar fashion to the first compute node in the example architectures section. Each additional compute node requires a unique IP address.

Install and configure components

Note : Default configuration files vary by distribution. You might need to add these sections and options rather than modifying existing sections and options. Also, an ellipsis (...) in the configuration snippets indicates potential default configuration options that you should retain.

1. Install the packages:

```
# apt-get install nova-compute
```

2. Edit the /etc/nova/nova.conf file and complete the following actions:

   – In the [DEFAULT] and [oslo_messaging_rabbit] sections, configure RabbitMQ message queue access:

```
[DEFAULT]
...
rpc_backend = rabbit

[oslo_messaging_rabbit]
...
rabbit_host = controller
rabbit_userid = openstack
rabbit_password = RABBIT_PASS
```

Replace RABBIT_PASS with the password you chose for the openstack account in RabbitMQ.

   – In the [DEFAULT] and [keystone_authtoken] sections, configure Identity service access:

```
[DEFAULT]
...
auth_strategy = keystone

[keystone_authtoken]
...
auth_uri = http://controller:5000
auth_url = http://controller:35357
memcached_servers = controller:11211
auth_type = password
project_domain_name = default
user_domain_name = default
project_name = service
username = nova
password = NOVA_PASS
```

Replace NOVA_PASS with the password you chose for the nova user in the Identity service.

➡ **Note :** Comment out or remove any other options in the [keystone_authtoken] section.

   – In the [DEFAULT] section, configure the my_ip option:

```
[DEFAULT]
...
my_ip = MANAGEMENT_INTERFACE_IP_ADDRESS
```

   – Replace MANAGEMENT_INTERFACE_IP_ADDRESS with the IP address of the management network interface on your compute node, typically 10.0.0.31 for the first node in the example architecture.

   – In the [DEFAULT] section, enable support for the Networking service :

```
[DEFAULT]
...
use_neutron = True
firewall_driver = nova.virt.firewall.NoopFirewallDriver
```

➡ **Note :** By default, Compute uses an internal firewall service. Since Networking includes a firewall service, you must disable the Compute firewall service by using the nova.virt.firewall.NoopFirewallDriver firewall driver.

   – In the [vnc] section, enable and configure remote console access:

```
[vnc]
...
enabled = True
vncserver_listen = 0.0.0.0
vncserver_proxyclient_address = $my_ip
novncproxy_base_url = http://controller:6080/vnc_auto.html
```

   – The server component listens on all IP addresses and the proxy component only listens on the management interface IP address of the compute node.

   – The base URL indicates the location where you can use a web browser to access remote consoles of instances on this compute node.

➡ **Note :** If the web browser to access remote consoles resides on a host that cannot resolve the controller hostname, you must replace controller with the management interface IP address of the controller node.

In the [glance] section, configure the location of the Image service API:

```
[glance]
...
api_servers = http://controller:9292
```

   – In the [oslo_concurrency] section, configure the lock path:

```
[oslo_concurrency]
...
lock_path = /var/lib/nova/tmp
```

- Due to a packaging bug, remove the logdir option from the [DEFAULT] section.
- Finalize installation

1. Determine whether your compute node supports hardware acceleration for virtual machines:

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

- If this command returns a value of one or greater, your compute node supports hardware acceleration which typically requires no additional configuration.
- If this command returns a value of zero, your compute node does not support hardware acceleration and you must configure libvirt to use QEMU instead of KVM.
- Edit the [libvirt] section in the /etc/nova/nova-compute.conf file as follows:

```
[libvirt]
...
virt_type = qemu
```

2. Restart the Compute service:

```
# service nova-compute restart
```

3.12 Deploying and Utilizing OpenStack in Production Environments, Building a Production Environment

This appendix describes an example production environment for a working OpenStack-Ansible (OSA) deployment with high availability services.

This example environment has the following characteristics:

- Three infrastructure (control plane) hosts
- Two compute hosts
- One NFS storage device
- One log aggregation host
- Multiple Network Interface Cards (NIC) configured as bonded pairs for each host.
- Full compute kit with the Telemetry service (ceilometer) included, with NFS configured as a storage back end for the Image (glance), and Block Storage (cinder) services.
- Internet access via the router address 172.29.236.1 on the Management Network.

Host and Service Layout - Production Ecvironment

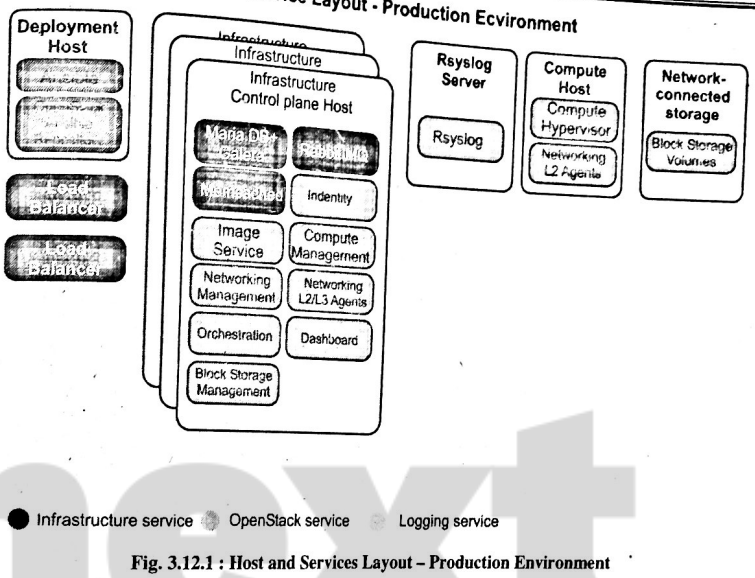

Fig. 3.12.1 : Host and Services Layout – Production Environment

Deployment configuration

Environment layout

The /etc/openstack_deploy/openstack_user_config.yml file defines the environment layout.

The following configuration describes the layout for this environment.

```
---
cidr_networks:
 container: 172.29.236.0/22
 tunnel: 172.29.240.0/22
 storage: 172.29.244.0/22

used_ips:
 - "172.29.236.1,172.29.236.50"
 - "172.29.240.1,172.29.240.50"
 - "172.29.244.1,172.29.244.50"
 - "172.29.248.1,172.29.248.50"

global_overrides:
 internal_lb_vip_address: 172.29.236.9
```

```
#
# The below domain name must resolve to an IP address
# in the CIDR specified in haproxy_keepalived_external_vip_cidr.
# If using different protocols (https/http) for the public/internal
# endpoints the two addresses must be different.
#
external_lb_vip_address: openstack.example.com
tunnel_bridge: "br-vxlan"
management_bridge: "br-mgmt"
provider_networks:
  - network:
      container_bridge: "br-mgmt"
      container_type: "veth"
      container_interface: "eth1"
      ip_from_q: "container"
      type: "raw"
      group_binds:
        - all_containers
        - hosts
      is_container_address: true
      is_ssh_address: true
  - network:
      container_bridge: "br-vxlan"
      container_type: "veth"
      container_interface: "eth10"
      ip_from_q: "tunnel"
      type: "vxlan"
      range: "1:1000"
      net_name: "vxlan"
      group_binds:
        - neutron_linuxbridge_agent
  - network:
      container_bridge: "br-vlan"
      container_type: "veth"
```



```
      container_interface: "eth12"
      host_bind_override: "eth12"
      type: "flat"
      net_name: "flat"
      group_binds:
        - neutron_linuxbridge_agent
  - network:
      container_bridge: "br-vlan"
      container_type: "veth"
      container_interface: "eth11"
      type: "vlan"
      range: "1:1"
      net_name: "vlan"
      group_binds:
        - neutron_linuxbridge_agent
  - network:
      container_bridge: "br-storage"
      container_type: "veth"
      container_interface: "eth2"
      ip_from_q: "storage"
      type: "raw"
      group_binds:
        - glance_api
        - cinder_api
        - cinder_volume
        - nova_compute

###
### Infrastructure
###

# galera, memcache, rabbitmq, utility
shared-infra_hosts:
  infra1:
```

```
  ip: 172.29.236.11
 infra2:
  ip: 172.29.236.12
 infra3:
  ip: 172.29.236.13

# repository (apt cache, python packages, etc)
repo-infra_hosts:
 infra1:
  ip: 172.29.236.11
 infra2:
  ip: 172.29.236.12
 infra3:
  ip: 172.29.236.13

# load balancer
# Ideally the load balancer should not use the Infrastructure hosts.
# Dedicated hardware is best for improved performance and security.
haproxy_hosts:
 infra1:
  ip: 172.29.236.11
 infra2:
  ip: 172.29.236.12
 infra3:
  ip: 172.29.236.13

# rsyslog server
log_hosts:
 log1:
  ip: 172.29.236.14

###
### OpenStack
###
```



```
# keystone
identity_hosts:
 infra1:
  ip: 172.29.236.11
 infra2:
  ip: 172.29.236.12
 infra3:
  ip: 172.29.236.13

# cinder api services
storage-infra_hosts:
 infra1:
  ip: 172.29.236.11
 infra2:
  ip: 172.29.236.12
 infra3:
  ip: 172.29.236.13

# glance
# The settings here are repeated for each infra host.
# They could instead be applied as global settings in
# user_variables, but are left here to illustrate that
# each container could have different storage targets.
image_hosts:
 infra1:
  ip: 172.29.236.11
  container_vars:
   limit_container_types: glance
   glance_nfs_client:
    - server: "172.29.244.15"
      remote_path: "/images"
      local_path: "/var/lib/glance/images"
      type: "nfs"
```

```
    options: "_netdev,auto"
infra2:
 ip: 172.29.236.12
 container_vars:
  limit_container_types: glance
  glance_nfs_client:
   - server: "172.29.244.15"
     remote_path: "/images"
     local_path: "/var/lib/glance/images"
     type: "nfs"
     options: "_netdev,auto"
infra3:
 ip: 172.29.236.13
 container_vars:
  limit_container_types: glance
  glance_nfs_client:
   - server: "172.29.244.15"
     remote_path: "/images"
     local_path: "/var/lib/glance/images"
     type: "nfs"
     options: "_netdev,auto"

# nova api, conductor, etc services
compute-infra_hosts:
 infra1:
  ip: 172.29.236.11
 infra2:
  ip: 172.29.236.12
 infra3:
  ip: 172.29.236.13

 # heat
 orchestration_hosts:
  infra1:
```



```
   ip: 172.29.236.11
  infra2:
   ip: 172.29.236.12
  infra3:
   ip: 172.29.236.13

# horizon
dashboard_hosts:
 infra1:
  ip: 172.29.236.11
 infra2:
  ip: 172.29.236.12
 infra3:
  ip: 172.29.236.13

# neutron server, agents (L3, etc)
network_hosts:
 infra1:
  ip: 172.29.236.11
 infra2:
  ip: 172.29.236.12
 infra3:
  ip: 172.29.236.13

# ceilometer (telemetry API)
metering-infra_hosts:
 infra1:
  ip: 172.29.236.11
 infra2:
  ip: 172.29.236.12
 infra3:
  ip: 172.29.236.13

# aodh (telemetry alarm service)
```

```
metering-alarm_hosts:
 infra1:
  ip: 172.29.236.11
 infra2:
  ip: 172.29.236.12
 infra3:
  ip: 172.29.236.13

# gnocchi (telemetry metrics storage)
metrics_hosts:
 infra1:
  ip: 172.29.236.11
 infra2:
  ip: 172.29.236.12
 infra3:
  ip: 172.29.236.13

# nova hypervisors
compute_hosts:
 compute1:
  ip: 172.29.236.16
 compute2:
  ip: 172.29.236.17

 # ceilometer compute agent (telemetry)
 metering-compute_hosts:
  compute1:
   ip: 172.29.236.16
  compute2:
   ip: 172.29.236.17

 # cinder volume hosts (NFS-backed)
 # The settings here are repeated for each infra host.
 # They could instead be applied as global settings in
```



```
# user_variables, but are left here to illustrate that
# each container could have different storage targets.
storage_hosts:
 infra1:
  ip: 172.29.236.11
  container_vars:
   cinder_backends:
    limit_container_types: cinder_volume
    nfs_volume:
     volume_backend_name: NFS_VOLUME1
     volume_driver: cinder.volume.drivers.nfs.NfsDriver
     nfs_mount_options: "rsize=65535,wsize=65535,timeo=1200,actimeo=120"
     nfs_shares_config: /etc/cinder/nfs_shares
     shares:
      - ip: "172.29.244.15"
        share: "/vol/cinder"
 infra2:
  ip: 172.29.236.12
  container_vars:
   cinder_backends:
    limit_container_types: cinder_volume
    nfs_volume:
     volume_backend_name: NFS_VOLUME1
     volume_driver: cinder.volume.drivers.nfs.NfsDriver
     nfs_mount_options: "rsize=65535,wsize=65535,timeo=1200,actimeo=120"
     nfs_shares_config: /etc/cinder/nfs_shares
     shares:
      - ip: "172.29.244.15"
        share: "/vol/cinder"
infra3:
 ip: 172.29.236.13
 container_vars:
  cinder_backends:
   limit_container_types: cinder_volume
```

```
nfs_volume:
 volume_backend_name: NFS_VOLUME1
 volume_driver: cinder.volume.drivers.nfs.NfsDriver
 nfs_mount_options: "rsize=65535,wsize=65535,timeo=1200,actimeo=120"
 nfs_shares_config: /etc/cinder/nfs_shares
 shares:
  - ip: "172.29.244.15"
    share: "/vol/cinder"
```

☞ Environment customizations

- The optionally deployed files in /etc/openstack_deploy/env.d allow the customization of Ansible groups. This allows the deployer to set whether the services will run in a container (the default), or on the host (on metal).
- For this environment, the cinder-volume runs in a container on the infrastructure hosts. To achieve this, implement /etc/openstack_deploy/env.d/cinder.yml with the following content:

```
---
# This file contains an example to show how to set
# the cinder-volume service to run in a container.
#
# Important note:
# When using LVM or any iSCSI-based cinder backends, such as NetApp with
# iSCSI protocol, the cinder-volume service *must* run on metal.
# Reference: https://bugs.launchpad.net/ubuntu/+source/lxc/+bug/1226855

container_skel:
 cinder_volumes_container:
  properties:
   is_metal: false
```

☞ User variables

- The /etc/openstack_deploy/user_variables.yml file defines the global overrides for the default variables.
- For this environment, implement the load balancer on the infrastructure hosts. Ensure that keepalived is also configured with HAProxy in /etc/openstack_deploy/user_variables.yml with the following content.

```
---
# This file contains an example of the global variable overrides
# which may need to be set for a production environment.
```



```
## Load Balancer Configuration (haproxy/keepalived)
haproxy_keepalived_external_vip_cidr: "1.2.3.4/32"
haproxy_keepalived_internal_vip_cidr: "172.29.236.0/22"
haproxy_keepalived_external_interface: ens2
haproxy_keepalived_internal_interface: br-mgmt
```

3.14 Application Orchestration using OpenStack Heat

GQ. 3.14.1 Explain Heat orchestration in OpenStack

UQ. 3.14.2 OpenStack orchestration is performed using ____ components. (MU - April 19, 1 Mark)

- Heat is the main project of the OpenStack orchestration program. It allows users to describe deployments of complex cloud applications in text files called *templates*. These templates are then parsed and executed by the Heat engine.
- Heat was born as the counterpart to the CloudFormation service in AWS. It accepts AWS templates and provides a compatible API, but in recent OpenStack releases it has also began to grow outside of the shadow of CloudFormation, providing a nicer template syntax (the Heat Orchestration Template, or HOT) and new features not supported by its competitor.

☞ Installing the Heat Client

- As is common with all OpenStack services, low-level access to Heat is available through a REST API. In most cases, however, working with a client is more convenient. There are two official Heat clients; a stand-alone command line client and a web-based client included with Horizon, the OpenStack dashboard project.
- All the examples I present in this series of articles use the command line client, which you have to install on your computer. I am not going to show how to use the web-based client, but you should have no trouble figuring it out on your own as it is very simple.
- The following commands create a Python virtual environment and install the Heat command line client in it, assuming you use bash or similar command prompt:

```
$ virtualenv venv
$ source venv/bin/activate
(venv) $ pip install python-heatclient
```

- For those working on Windows, the commands are slightly different:

```
$ virtualenv venv
$ venv\Scripts\activate
(venv) $ pip install python-heatclient
```

- Note that the above commands assume you have Python and virtualenv already installed on your system.

- To verify that heat was installed successfully, just run the client without arguments to see its help message :

```
(venv) $ heat
usage: heat [--version] [-d] [-v] [-k] [--os-cacert <ca-certificate>]
        [--cert-file CERT_FILE] [--key-file KEY_FILE] [--ca-file CA_FILE]
        [--api-timeout API_TIMEOUT] [--os-username OS_USERNAME]
        [--os-password OS_PASSWORD] [--os-tenant-id OS_TENANT_ID]
        [--os-tenant-name OS_TENANT_NAME] [--os-auth-url OS_AUTH_URL]
        [--os-region-name OS_REGION_NAME] [--os-auth-token OS_AUTH_TOKEN]
        [--os-no-client-auth] [--heat-url HEAT_URL]
        [--heat-api-version HEAT_API_VERSION]
        [--os-service-type OS_SERVICE_TYPE]
        [--os-endpoint-type OS_ENDPOINT_TYPE] [--include-password]
        <subcommand> ...
```

- Like the other OpenStack command line clients, the heat client needs to have access to your account credentials, which you normally have in a OPENRC file.
- For the examples in this article I assume that you have imported your OPENRC credentials into the environment, so that there is no need to include credentials as command line arguments.

A Basic Heat Template

You are probably anxious to see how Heat works, so let's dive right into it. Below you can see a very simple HOT template :

```
heat_template_version: 2013-05-23

description: Simple template to deploy a single compute instance

resources:
  my_instance:
    type: OS::Nova::Server
    properties:
      image: cirros-0.3.3-x86_64
      flavor: m1.small
      key_name: my_key
      networks:
        - network: private-net
```


As you can see in the example, HOT templates are written as structured YAML text files. This particular example contains three top-level sections :

- heat_template_version is a mandatory section that is used to specify the version of the template syntax that is used.
- Most templates you are going to see out there will have 2013-05-23 as version, the first release. There is also a newer version labeled 2014-10-16 and introduced with the Juno release that contains a few minor changes and additions.
- description is optional, and it is used to provide a description of what the template does.
- resources is the most important section in a template, because this is where the different components are defined.
- In this first example, the only resource is called my_instance, and it is declared with type OS::Nova::Server, which is the type of a Nova compute instance. The properties sub-section identifies which image, flavor, public key and private network to use for the instance.
- Are you ready to launch this template? Copy/paste the above text into your favorite text editor, edit the image, flavor, key and private network names to match your OpenStack installation, and save the file as heat_1a.yaml.

> **Note :** If you are working with OpenStack Havana or Icehouse, then the private network needs to be specified as an id instead of a name. You can find the id of your network using the nova net-list or neutron net-list commands. In the Juno release both the name and the id are supported.

- Once you have the template on disk, you can use the following command to create a stack from the template:

```
(venv) $ heat stack-create my_first_stack -f heat_1a.yaml
+--------+---------------+-----------------+---------------+
| id     | stack_name    | stack_status    | creation_time |
+--------+---------------+-----------------+---------------+
| ...    | my_first_stack | CREATE_IN_PROGRESS | 2014-11-05T18:10:40Z |
+--------+---------------+-----------------+---------------+
```

- Heat then starts a background job that instantiates the resources declared in the template. In this case, that resource is just a compute instance. To query the status of this job, use the following command:

```
(venv) $ heat stack-show my_first_stack
```

- The output shows all the information for this stack, including its status, which will eventually be CREATE_COMPLETE (or CREATE_FAILED if there was an error).

Template Parameters and Outputs

- The template I presented in the previous section is extremely simple and not very useful. It is actually not very convenient to have to edit the template to match a particular OpenStack installation.
- Let's look at an improved version :

```
heat_template_version: 2013-05-23

description: Simple template to deploy a single compute instance

parameters:
  image:
    type: string
    label: Image name or ID
    description: Image to be used for compute instance
    default: cirros-0.3.3-x86_64
  flavor:
    type: string
    label: Flavor
    description: Type of instance (flavor) to be used
    default: m1.small
  key:
    type: string
    label: Key name
    description: Name of key-pair to be used for compute instance
    default: my_key
  private_network:
    type: string
    label: Private network name or ID
    description: Network to attach instance to.
    default: private-net

resources:
  my_instance:
    type: OS::Nova::Server
    properties:
```



```
    image: { get_param: image }
    flavor: { get_param: flavor }
    key_name: { get_param: key }
    networks:
      - network: { get_param: private_network }

outputs:
  instance_ip:
    description: IP address of the instance
    value: { get_attr: [my_instance, first_address] }
```

- This new version of the template adds two new top-level sections:
  - parameters is used to declare a list of inputs that need to be provided by the user.
  - outputs defines what attributes of the stack to export after it is deployed.
- By using a parameters section, a template can be made generic. Each parameter is given a name and a type and, optionally, a description and a default value.
- The get_param function is then used to insert parameter values into resource properties. Looking at the other side, the get_attr function is used in the outputs section to extract desired attributes of the resources included in the stack.
- To try this new template, save it as file heat_1b.yaml and launch it as shown before. Unless your system is identical to mine, you are probably going to get an error, because the parameter defaults that I defined will likely not match your OpenStack installation.
- However, since these settings are now parameters, you can specify appropriate values for your environment in the stack-create command without having to edit the template file.

For example:

```
(venv) $ heat stack-create second_stack -f heat_1b.yaml -P "key=my_key_name;image=Trusty"
```

- In this example, the key parameter is set to "my_key_name" and the image parameter is set to "Trusty", so those values will be used for this instantiation of the stack.
- For any parameters not included in the -P option, the defaults are used, which applies to flavor and private_network in this example.
- Note that parameters that do not have a default value defined must be included in the stack-create command, so it is a good idea to define defaults whenever possible.

- Once the stack is created, the stack-show command includes the attributes requested in the outputs section :

```
(venv) $ heat stack-show second_stack
+---------------------+--------------------------------------------------+
| Property            | Value                                            |
+---------------------+--------------------------------------------------+
| ...                 |                                                  |
| ...                 |                                                  |
| outputs             | [                                                |
|                     |   {                                              |
|                     |     "output_value": "10.10.10.72",               |
|                     |     "description": "IP address of the instance", |
|                     |     "output_key": "instance_ip"                  |
|                     |   }                                              |
|                     | ]                                                |
| ...                 |                                                  |
| ...                 |                                                  |
+---------------------+--------------------------------------------------+
```

Chapter Ends...

❑❑❑

LAB MANUAL

Practical 1

- **Aim : Study and implementation of Infrastructure as a Service.**
- **Theory :**

☞ Infrastructure as a Service (IAAS)

- IaaS, as the term suggests, is a way of providing cloud computing infrastructure like virtual machines, servers, storage drives, operating systems and networks, which is also on same condition i.e. an on-demand service as like SaaS.
- Instead of purchasing servers or developing software, clientscan purchase those resources as a fully outsourced service based on their need.
- "Public cloud" is considered as an infrastructure that consists of shared resources, based on a self-service over the Internet.
- In one word, it is the only layer of the cloud where the customer gets the platform for their organization to outsource IT infrastructure on a pay-per-use basis.
- The "public cloud" uses infrastructure consisting of shared resources.
- In another word, it is the only layer of the cloud where the client gets the platform for their organization to outsource the IT infrastructure on a pay-per-use basis.

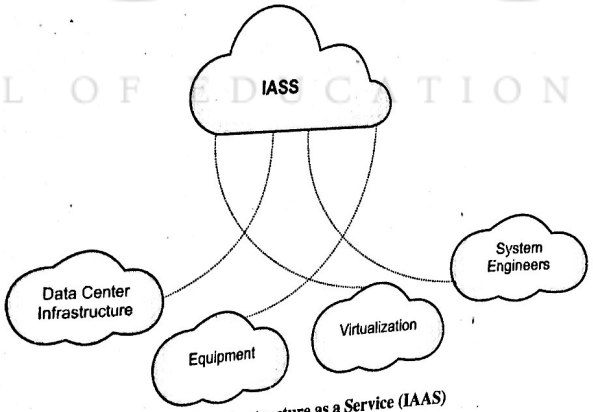

Fig. 1 : Infrastructure as a Service (IAAS)

☞ IaaS provides users with :

- Load balancers
- Disk storage via virtual machines
- Software Packages
- IP address
- VLANs