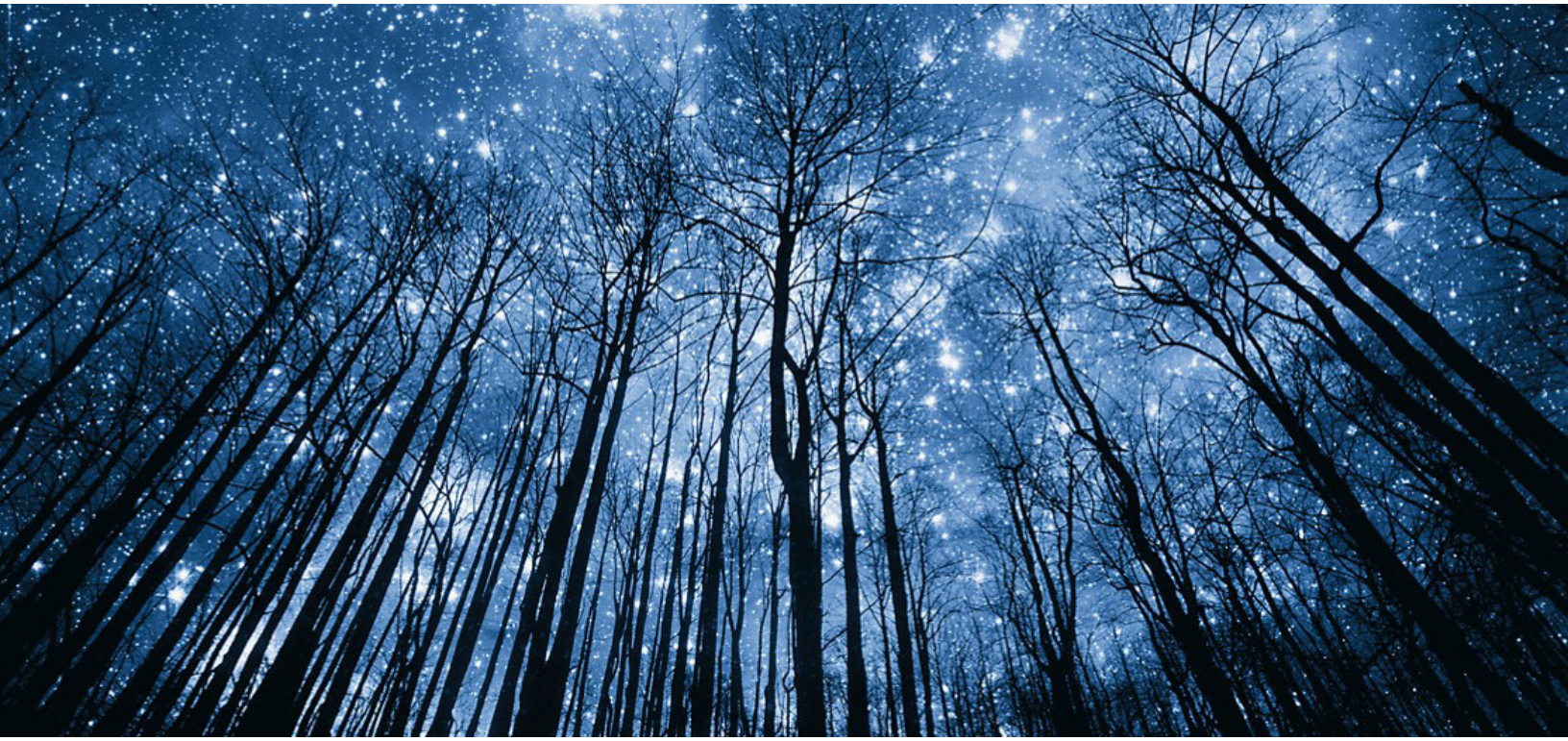
DELL Technologies

# GETTING STARTED WITH THE OPEN-SOURCE CLOUD




## Luis Cardenas
Senior Systems Engineer,
Solutions Architect
Dell Technologies

## Kartikeya Chauhan
Associate Sales Engineer Analyst
Dell Technologies


DELL Technologies
Proven Professional

Knowledge Sharing Article

# Contents

## Introduction

The cloud is an area of focus that many organizations have come to adopt in their technology strategy. But what exactly is the cloud? The National Institute of Standards and Technology (NIST) define cloud computing as a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.[1] Many organizations utilize one or more of these services and resources today.

There are three main cloud models: Public, Private and Hybrid. At a very high level, the differences between these models are the sharing of resources among consumers. Public cloud models typically have hundreds or thousands of different companies or individual public users sharing the same physical infrastructure and resources. The private cloud generally serves a single entity or company hosted on dedicated infrastructure and does not share resources with users or groups outside of that company. In a hybrid cloud model, an organization or user extends network connectivity between public and private clouds to allow for moving workloads between the two cloud infrastructures. Though there are differences between these cloud models, the foundation to them all is software. There are many software packages available, both commercial and open-source, to help an organization on their cloud journey.

In this paper, we will provide a guide to deploying your own private cloud based on the open-source projects from the OpenStack framework.

## The OpenStack Framework

OpenStack is an open-source cloud computing platform that enables organizations to build and manage cloud computing environments. It is made up of a collection of interrelated services that work together to provide infrastructure as a service (IaaS). OpenStack provides a wide range of services, including:

- **Compute:** Enables users to create and manage virtual machines (VMs) and other compute resources.
- **Networking:** Provides networking and connectivity for VMs and other compute resources.
- **Storage:** Provides persistent storage for VMs and other compute resources.
- **Identity and Access Management (IAM):** Provides a central location for managing user accounts, groups, and permissions.
- **Image:** Enables users to create, store, and manage disk images for VMs and other compute resources.
- **Dashboard:** The web-based user interface for managing OpenStack resources.

Its core purpose is to be highly modular and scalable, making it suitable for use in a wide range of cloud computing environments, including public and private clouds. The platform is widely used by organizations of all sizes, including Fortune 500 companies.[2] In this paper we will guide a user through a basic installation of the platform. Before diving into the installation, let us first review some of the history of how OpenStack came to be.

OpenStack was founded in 2010 by Rackspace Hosting and NASA as a joint project to build an open-source cloud computing platform. The goal was to provide a scalable and flexible infrastructure that could be used by organizations to build and manage their own cloud computing environments. The first version of OpenStack, known as "Austin," was released in October 2010. "Austin" included Compute, Object Storage, and Image Service components. Over the next few years, OpenStack continued to evolve and add new features, including support for networking, identity, and access management. The OpenStack Foundation was created a few years later in 2013 to oversee the development and growth of the project. The foundation is a non-profit organization that is responsible for promoting the use of OpenStack and its open-source software community.

Open-source software is software that is made available to the public with a license that allows users to freely use, modify, and distribute the software. This type of licensing model encourages collaboration and participation from a community of developers, who can contribute to the development and improvement of the software. One of the key benefits of open-source software is that it is typically developed and maintained by a large, active community of volunteers. This can lead to faster development and a more robust product since there are many people working on the software and contributing their expertise. Open-source software is also more transparent, as the source code is available for anyone to view and modify--the very can help to build trust with users, as they can see exactly how the software works and what it does.

OpenStack and more broadly, open-source software have grown in popularity and is now used by a wide range of organizations around the world. It has become a major player in the cloud computing market, with many organizations choosing to use OpenStack as the basis for their private cloud infrastructure. Open-source software and the associated licensing models continues to gain support as technology becomes a critical factor to businesses in every industry. There are many different open-source licenses, each with its own terms and conditions. Some common open-source licenses include the GNU General Public License (GPL), the MIT License, and the Apache License (OpenStack is licensed under the Apache License). It is important for users of open-source software to understand the terms of the license and how they can use the software in accordance with the license.

Open-source licenses and commercial licenses are two different types of licensing models that are used to govern the use of software. Open-source licenses are designed to allow users to freely use, modify, and distribute the software. These licenses typically have minimal restrictions on use and allow users to do almost anything they want with the software if they follow the terms of the license. Commercial licenses, on the other hand, are typically paid licenses that grant users the right to use the software for a specific purpose or in a specific way. These licenses may have more restrictive terms and may prohibit users from modifying or distributing the software, which has its own advantages and disadvantages. There are several key differences between open-source licenses and commercial licenses:

- **Freedom:** Open-source licenses generally grant users more freedom to use, modify, and distribute the software, while commercial licenses may be more restrictive.
- **Cost:** Open-source licenses are typically free to use, while commercial licenses usually require users to pay a fee.
- **Modification:** Open-source licenses generally allow users to modify the software, while commercial licenses may prohibit or limit modification.
- **Distribution:** Open-source licenses typically allow users to distribute the software, while commercial licenses may restrict distribution.

Examples of commercially available cloud software stacks are VMware's Cloud Foundation suite and Microsoft's Azure Stack. An example of another open-source cloud stack is OpenNebula. It is important for users of both open-source and commercial software to understand the terms of the license and to use the software in accordance with the terms of the license.

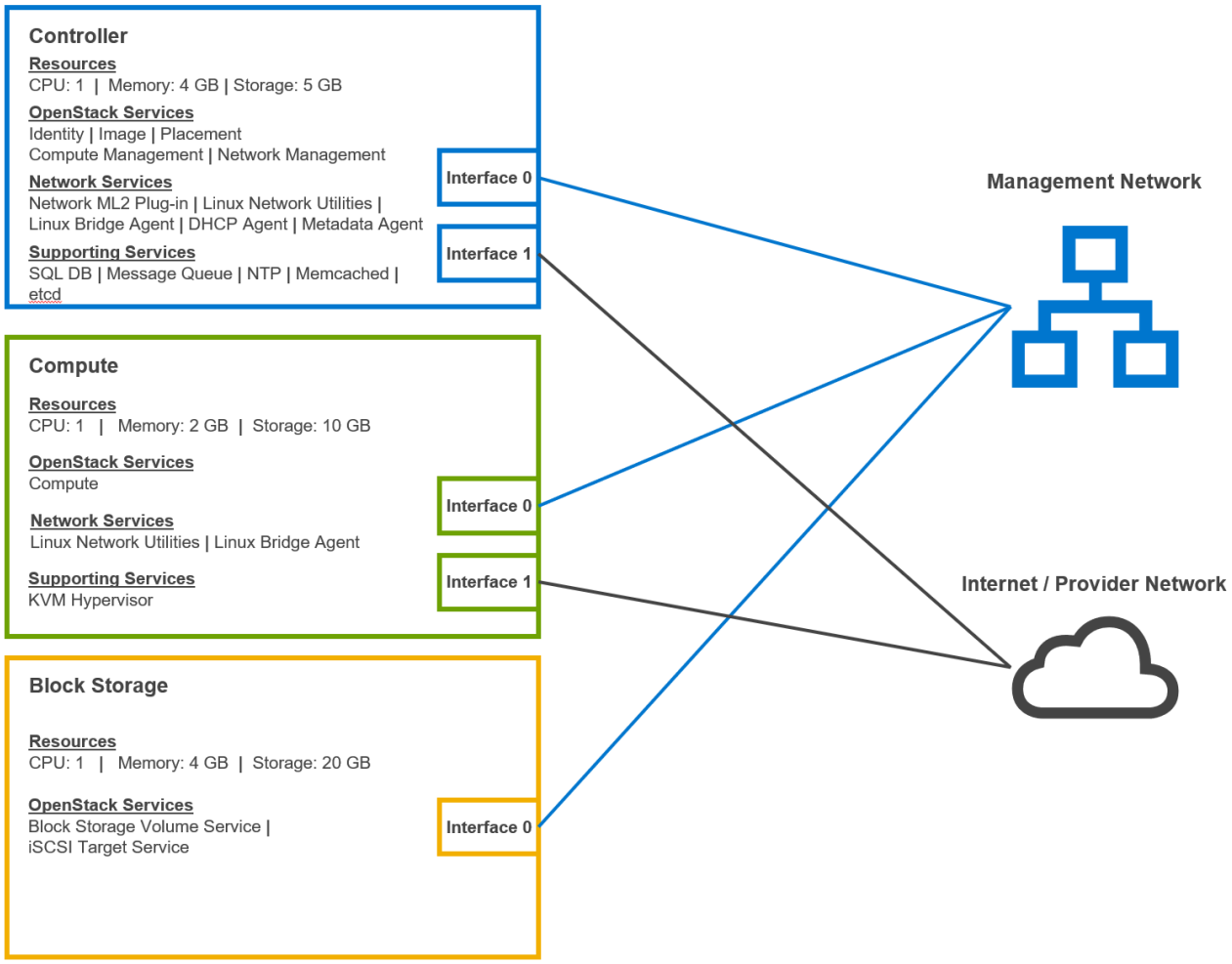
## Getting Started with OpenStack

The OpenStack framework consist of over two dozen services, also known as projects. These services communicate with each other through API calls which are routed via a message broker with the services' states kept in a database. Users can choose from several supported databases and AQMP message brokers including RabbitMQ, MySQL and MariaDB. OpenStack also relies on Memcached in conjunction with the Identity service to provide token caching and etcd to support ancillary functions. Together, these services form an Infrastructure-as-a-Service foundation for your private cloud.

As part of this tutorial, we will review a minimal viable product deployment of OpenStack on Ubuntu 22.04. We provide here a distilled set of instructions from the official OpenStack installation guide found at https://docs.openstack.org/install-guide/. This guide can and should be referenced for the most up to date installation procedures, and configuration options, as it is continuously updated. We will focus on the deployment of a core subset of the available services:

- Identity service (Keystone)
- Image service (Glance)
- Placement service (Placement)
- Compute service (Nova)
- Network service (Neutron)
- Block storage service (Cinder)
- Dashboard WebUI service (Horizon)

These core services, along with supporting packages, will be installed on nodes which can be either physical or virtual machines. We will utilize three nodes designated as the Controller, Compute and Block Storage nodes. We will focus on deploying the environment on the Provider Network. This is a basic network architecture utilizing bridging to provide connectivity between nodes, VM's and storage. This network architecture is limited in the services it can support but will allow for this proof of concept. Advanced network architectures from the OpenStack maintainers and Linux distributions are recommended for use in production environments and is referred to as the Self-Service Network. The following diagram is a high-level overview of the nodes we will be deploying and the associated services:




### Configure Network Connectivity

We will begin configuring network interfaces on the three nodes. Apply a static IP address to the first interface by modifying the `/etc/netplan/01-network-manager-all.yaml` file similar to below. In this example we use the 192.168.1.0/24 network, however you may choose a different network. We will also need the second interface for the Provider Network defined without an IP address. Note you will need to reference the appropriate interface and IP address for the node you are updating. Apply the changes in the yaml file using the `netplan apply` command.

Controller Node first interface: 192.168.1.10

Compute Node first interface: 192.168.1.20

Block Storage Node first interface: 192.168.1.30

**Example:**

```
network:
    Version: 2
```

```
Renderer: NetworkManager
ethernets:
   DEVICE_NAME:
      ens1:
       dhcp4: no
       addresses: [192.168.1.10/24]
       gateway: 192.168.1.1
       Nameservers:
            Addresses: [8.8.4.4]
      ens2: {}
```

Next, update the `/etc/hosts` file on each node to contain the entries for the Controller, Compute, and Block Storage nodes.

```
# controller
192.168.1.10 controller
# compute
192.168.1.20 compute
# blockStorage
192.168.1.30 blockStorage
```

Verify connectivity from the Controller and Compute nodes to the Internet using a ping test to www.openstack.org. You may need to update firewall rules on the host to ensure the node can reach external sites. Similarly test the Controller, Compute and Block Storage nodes can ping each other's management interface IP address.

Install and configure Chrony on the Controller node to provide the NTP service:

```
# apt install chrony
```

This will create the `/etc/chrony/chrony.conf` file and will need to be updated to allow the Compute and Block Storage nodes to connect. Update this file to include the management network:

```
allow 192.168.1.0/24
```

Install Chrony on the Compute and Block Storage nodes:

```
# apt install chrony
```

Finally, edit the `/etc/chrony/chrony.conf` file on those two nodes to comment out all entries and include reference to the Controller for time synchronization. Then reboot to ensure changes take effect.

```
server controller iburst

# reboot
```

We can now verify the Controller node is connecting to the defined NTP pools in the chrony.conf file, and that the Compute and Block Storage nodes are connecting to the Controller node for time synchronization by issuing the below command on each node:

```
# chronyc sources
```

At this point, we should have the base connectivity between all nodes up and running and can move on to setting up the OpenStack packages and their supporting services.

## Install OpenStack Services

Next, we will install the OpenStack services on the nodes. To download the packages, we will need to update the node's repository to include the OpenStack release archive. As we are trying to install the OpenStack Zed release packages, you will need to ensure the `software-properties-common` package is at release 0.99.22.1 or later. On each node, add the archive for the OpenStack Zed release to the Ubuntu repository:

```
# add-apt-repository cloud-archive:zed
```

As the services found in the repository are frequently updated and may require adjustments to installation, we will provide direct links to the installation procedure for the individual OpenStack services on Ubuntu for the nodes described here. It is highly recommended to use the latest documentation set for the OpenStack release you are using.

**Controller Node**

Identity Service - https://docs.openstack.org/keystone/zed/install/index-ubuntu.html

Image Service - https://docs.openstack.org/glance/zed/install/install-ubuntu.html

Placement Service - https://docs.openstack.org/placement/zed/install/install-ubuntu.html

Compute Service - https://docs.openstack.org/nova/zed/install/controller-install-ubuntu.html

Networking Service - https://docs.openstack.org/neutron/zed/install/controller-install-ubuntu.html

Block Storage Service - https://docs.openstack.org/cinder/zed/install/cinder-controller-install-ubuntu.html

Dashboard Service - https://docs.openstack.org/horizon/zed/install/install-ubuntu.html

**Compute Node**

Compute Service - https://docs.openstack.org/nova/zed/install/compute-install-ubuntu.html

Networking Service - https://docs.openstack.org/neutron/zed/install/compute-install-ubuntu.html

**Block Storage Node**

Block Storage Service - https://docs.openstack.org/cinder/zed/install/cinder-storage-install-ubuntu.html

Reboot the nodes to allow changes to take effect.

```
# reboot
```

## Install Supporting Services

Earlier in this guide we discussed OpenStack's use of supporting services such as a SQL database and AQMP message broker to facilitate some of the communication between nodes. The following steps will show installing MariaDB and RabbitMQ services on the Controller node:

```
# apt install mariadb-server python3-pymysql
```

Create and edit the `/etc/mysql/mariadb.conf.d/99-openstack.cnf` to set the bind address to the management interface on the Controller node:

```
[mysqld]
bind-address = 192.168.1.10
default-storage-engine = innodb
innodb_file_per_table = on
max_connections = 4096
collation-server = utf8_general_ci
character-set-server = utf8
```

Restart the SQL database service and set the database password

```
# service mysql restart
# mysql_secure_installation
```

Install the RabbitMQ message service, configure the openstack user and replace PASSWORD with a password of your choosing. Finally, grant permissions to the openstack user:

```
# apt install rabbitmq-server
```

```
# rabbitmqctl add_user openstack PASSWORD
# rabbitmqctl set_permissions openstack ".*" ".*" ".*"
```

Two additional services, memcached and etcd, help support OpenStack services. We will install these on the Controller node as well.

```
# apt install memcached python3-memcache
# apt install etcd
```

In the `/etc/memcached.conf` file, replace the IP address to the management interface IP address of the Controller node. Similarly, edit the `/etc/default/etcd` file and set the values for the below lines to also use the management interface IP address.

```
ETCD_INITIAL_CLUSTER="controller=http://192.168.1.10:2380"
ETCD_INITIAL_ADVERTISE_PEER_URLS="http://192.168.1.10:2380"
ETCD_ADVERTISE_CLIENT_URLS="http://192.168.1.10:2379"
ETCD_LISTEN_CLIENT_URLS="http://192.168.1.10:2379"
```

Reboot the nodes to allow changes to take effect.

```
# reboot
```

Once all nodes come back up, we should now have a proof-of-concept OpenStack environment We can test the creation of a VM and Block Storage device. At this point, we should now have an environment that we can test VM deployment and Block Storage creation.

## Deploy a Virtual Machine and Volume

Now that we have an OpenStack environment built, we can test creating a Virtual Machine and a Block Storage device. To do this, we will need to first create the virtual network the machine will be connected to. We will define this virtual network on the Provider Network that we set up earlier.

Elevate your privileges to gain access to OpenStack admin CLI commands,

```
$ . admin-openrc
```

and define the network:

```
$ openstack network create --share --external \
   --provider-physical-network provider \
   --provider-network-type flat provider
```

Edit the `/etc/neutron/plugins/ml2/ml2_conf.ini` file with the Provider network under the `[ml2_type_flat]` section:

```
[ml2_type_flat]
flat_networks = provider
```

Additionally, add the Provider network interface to the `[linux_bridge]` section of the `/etc/neutron/plugins/ml2/ linuxbridge_agent.ini` file, utilizing the appropriate interface name on the system:

```
[linux_bridge]
physical_interface_mappings = provider:ens2
```

Now, create the subnet of your choosing utilizing the Provider network. This will be our VM subnet. interface name on the system:

```
openstack subnet create --network provider \
--allocation-pool start=192.168.100.100,end=192.168.100.149 \
--dns-nameserver 8.8.4.4 --gateway 192.168.100.1 \
--subnet-range 192.168.100.0/24 provider
```

In this example, we have selected the network 192.168.100.0/24 to use for our VM's with an IP allocation pool of 50 addresses between 192.168.100.100 : 192.168.100.149. You may use a different network. Additionally, we are using the DNS server defined in our `/etc/resolv.conf` file. We are now ready to create the instance flavor, generate the keypair and update security group rules which will be needed to launch and access a VM.

Define the flavor we will use for testing purposes:

```
$ openstack flavor create --id 0 --vcpus 1 --ram 64 \

--disk 1 m1.nano
```

Elevate into the demo credentials and generate the keypair. Then add it to the OpenStack environment for use:

```
$ . demo-openrc

$ ssh-keygen -q -N ""

$ openstack keypair create --public-key ~/.ssh/id_rsa.pub mykey
```

Update the security group rules to allow ICMP traffic and SSH access:

```
$ openstack security group rule create --proto icmp default

$ openstack security group rule create --proto tcp --dst-port 22
\ default
```

You will now be set up to launch a VM instance on the Provider network using the flavor and keys we created. We will deploy a lightweight Linux image, CirrOS, for this test. You will need to replace the **`PROVIDER_NET_ID`** value with your environment's Provider network network_id. You can find this network_id value displayed during the `openstack subnet create` command we issued earlier, or by listing out the available networks and their network_id values via the `openstack network list` command.

```
$ openstack server create --flavor m1.nano --image cirros \
--nic net-id=PROVIDER_NET_ID --security-group default \
--key-name mykey provider-instance
```

List the newly created VM and access via SSH from the Controller node:

```
$ openstack server list
$ ssh cirros@192.168.100.101
```

We should now have a live VM and can proceed to create a block storage volume and attach it.

From the Controller node, create a 1 GB block storage volume named vol_01:

```
$ openstack volume create --size 1 vol_01
```

We can show the volume's status using the list command and attach it to the VM. Utilize the instance name and volume name you chose during the creation of those objects:

```
$ openstack volume list
$ openstack server add volume provider-instance vol_01
```

List the updated status of the volume. It should now show as attached to the `provider-instance` VM:

```
$ openstack volume list
```

You are now able to configure the newly attached block storage device using fdisk via the SSH connection to the CirrOS VM.

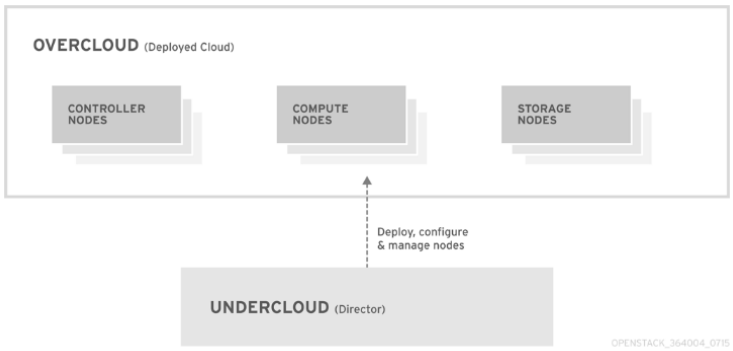
We have now successfully shown how to install a minimal viable product OpenStack environment utilizing three nodes. While this is a rudimentary approach to installation, we can see the complexities of manual installation in just a three-node environment. Enterprise production environments will utilize dozens or hundreds of nodes and more complex network architectures. As such, many of the popular Linux distributions like Ubuntu and Red Hat have developed automation for deployment of OpenStack and are the recommended deployment method. We will explore some of these deployment methods in our next section.

## Deployment

Now that we have a basic understanding of the core services for OpenStack and their manual installation, let us discuss the various methods to deploy it. We will provide a high-level understanding of the deployment options available and a general exploration of two of the most popular types: native deployment (TripleO) and containerized deployment using Koalla-Ansible playbooks. There are several ways to deploy OpenStack, depending on your specific needs and resources. The following are the most common approaches:

- **OpenStack distribution:** There are several companies that offer distributions of OpenStack, which include pre-packaged versions of the software that have been tested and configured for easy deployment. Some examples of OpenStack distributions include Red Hat OpenStack Platform, SUSE OpenStack Cloud, and Ubuntu OpenStack.
- **OpenStack-Ansible:** This is a project within the OpenStack community that provides a set of Ansible playbooks for deploying OpenStack. These playbooks can be used to automate the installation and configuration of an OpenStack cloud.
- **OpenStack-Helm:** This is a project within the OpenStack community that provides a set of helm charts for deploying OpenStack. Helm is a package manager for Kubernetes, and the OpenStack-Helm charts provide a way to deploy OpenStack services as Kubernetes pods.
- **Manual Installation of Services:** You can also install OpenStack manually by following the installation instructions provided in the OpenStack documentation, and which were provided in this document section 3. This can be a more complex approach and is generally not recommended for environments requiring scale.

Let us now understand the core methodologies pertaining to the native deployment using the TripleO method, using OpenStack's own cloud facilities as the foundation. It is designed to be a highly automated and scalable way to deploy OpenStack clouds, and it is particularly well-suited for large-scale deployments. A system comprised of the undercloud and the overcloud will need to be deployed in the following steps:




1. **Prepare the undercloud:** The undercloud is the OpenStack deployment that will be used to manage the overcloud, which is the target OpenStack cloud being deployed. The undercloud is typically deployed on a single physical host and includes the necessary OpenStack components and services for managing the overcloud.

2. **Deploy the overcloud:** The overcloud is the target OpenStack cloud that is being deployed using TripleO. It is deployed on one or more physical hosts, and it includes the necessary OpenStack components and services for running the cloud.

3. **Configure the overcloud:** Once the overcloud has been deployed, you can use the undercloud to configure the various OpenStack services and options in the

overcloud. This can be done using OpenStack's native configuration management tools, such as Heat or Ansible, or using custom scripts and tools.

4. **Manage the overcloud:** Once the overcloud is up and running, you can use the undercloud to manage and maintain it, including tasks such as upgrading the cloud software, adding new compute resources, and scaling the cloud up or down as needed.

The complete documentation is available for the TripleO deployment guide is available here https://docs.openstack.org/project-deploy-guide/tripleo-docs/latest/, for reference.

Finally, if the requirements suit production-ready containers which can easily deploy OpenStack, we can follow these steps using a set of Ansible playbooks (Koalla-Ansible). Here is a high-level overview of the process:

1. **Install ansible:** Before you can use the Koalla-ansible playbooks, you will need to install ansible on your system. Ansible is a configuration management and automation tool that is used to manage the various nodes in your OpenStack cloud.

2. **Download the Koalla-ansible repository:** You will need to download the Koalla-ansible repository from GitHub and extract it to a local directory on your system. This repository contains the ansible playbooks and other necessary files for deploying an OpenStack cloud.

3. **Configure the inventory:** The inventory is a file that specifies the hostnames and other details of the nodes in your OpenStack cloud. You will need to edit the inventory file to specify the hostnames and other details of your nodes.

4. **Configure the playbook variables:** The Koalla-ansible playbooks include a number of variables that you can use to customize the deployment of your OpenStack cloud. You will need to edit the playbook files and set the variables to the values that you want.

5. **Run the playbooks:** Once you have configured the inventory and playbook variables, you can run the playbooks to deploy your OpenStack cloud. This is typically done using the following command: `ansible-playbook -i <inventory-file> <playbook>.yml`

6. **Verify the deployment:** Once the playbooks have completed, you should verify that your OpenStack cloud has been deployed correctly. You can do this by logging in to the OpenStack dashboard and checking the status of the various OpenStack services.

You can refer to the documentation here: https://docs.openstack.org/project-deploy-guide/kolla-ansible/zed/ , for a complete list of steps.

## Lifecycle and Maintenance

**How do I upgrade?**

Like with any technology landscape, keeping current on release schedules and patches will ensure the best experience and uptime. New features, bug fixes and security patches are developed from the code base maintainers with periodic release to the public. Release upgrades for OpenStack are no different and should stay current with the stable versions available. Upgrades can be challenging and will require sufficient planning to execute. The complexities raise considerations to keep in mind:

- Cloud instances might be disrupted while upgrading. In case of mission critical deployments, this is a key concern.
- Prepare for rollback processes in case of failure while upgrading. Refer here for more details on how to rollback a failed upgrade https://docs.openstack.org/operations-guide/ops-upgrades.html#rolling-back-a-failed-upgrade
- There are some incompatibilities between versions, which need to be reviewed, so a proper consultation of release notes is paramount.
- Upgrading immediately to a newly released version can be risky, due to bugs, so a small-scale pre-testing is helpful to screen these issues before rolling out a wide update.

Here, we will discuss a basic upgrade process, using the two-node architecture. We will need to update previously discussed core services individually. Before we can proceed with upgrading the environment, we must clean up half-purged instances to ensure consistencies and perform a full backup. Then we must remove all previous release package repos and add target package repos for the desired version and finally update the repository database. The following is a high-level overview for upgrading these (in order):

**Controller Nodes**:

1. **Identity service (Keystone) upgrades:** To upgrade without downtime, make a full backup of your DB and suspend the keystone process on any node that will serve as the orchestrator for DB upgrades. Then, upgrade this node to the desired release without starting any keystone processes. Subsequently, verify deployment issues, update config files, and gradually update keystone configs on all nodes. Finally, upgrade all keystone nodes to the next release. A more detailed breakdown of the process is given here: https://docs.openstack.org/keystone/latest/admin/upgrading.html.
2. **Image service (Glance) upgrades:** This is as simple as running the command (where `[VERSION]` is the target version):

   ```
   $ glance-manage db upgrade [VERSION]
   ```

   For DB maintenance, to hard delete records (as glance usually performs soft-deletions), you can purge the database occasionally by using the glance-manage tool. Refer to https://docs.openstack.org/glance/latest/admin/db.html for a more detailed breakdown of the process

3. **Compute service (Nova) upgrades:** For compute upgrades, only nova services are upgraded here since there is no upgrade for the hypervisor. This means that subsequent upgrades can be followed to reduce downtime where current version (N) and the target version (N+1) co-exist briefly for a smooth rollover. Moreover, keep in mind that Nova does not support database downgrades, with Schema migrations and Data Migrations supported for database upgrades.
   An instructive step-by-step guide can be found here: https://docs.openstack.org/nova/latest/admin/upgrades.html.
   Make note that the Bare Metal services (Ironic) upgrades should always be done before updating the compute services, with cold and rolling upgrade methods discussed in the Ironic documentation here: https://docs.openstack.org/ironic/latest/admin/upgrade-guide.html.
4. **Networking service (Neutron) upgrades:** Here we roll out upgrades to Neutron Servers, which require the new database schema, and then to Neutron Agents. These software upgrades should not disrupt the data plane connectivity during the upgrade process. Finally, we validate the upgrade using Grenade ( https://github.com/openstack/grenade) through a series of tests.
   A complete process for Neutron Server and Agent upgrades can be found here: https://docs.openstack.org/neutron/latest/contributor/internals/upgrade.html.
5. **Block Storage service (Cinder) upgrades:** During this upgrade process, the database schema migrations are carried out smoothly so that both N and N+1 level codes can execute transactions against it. We upgrade the cinder-scheduler, cinder-volume services, and optional cinder-backup services with cinder-api services last. Finally, we ensure consistency of our upgraded repos. The upgrade process is very detailed and a thorough review of the latest documentation should be reviewed here: https://docs.openstack.org/cinder/latest/admin/upgrades.html.
6. Dashboard service (Horizon) upgrades: This upgrade is trivial and requires restarting the Apache HTTP service once the latest service has been installed.

## What happens when I encounter an issue or there are bugs?

There will always be bugs encountered in software and open-source software is no different. The reporting of bugs you will see is like a commercially distributed software package, in that your submission goes to the maintainers of the codebase. OpenStack uses StoryBoard stories for tracking the state of the bug following creation within Launchpad. Submitting bugs via Launchpad are documented extensively here: https://docs.openstack.org/project-team-guide/bugs.html

The page at https://wiki.openstack.org/wiki/DevQuickstart will cover the submissions steps in much more detail, but let us look at a quick overview:

1. Set up a Dev environment using http://devstack.org/, making sure you have a machine running at least ubuntu 11.10.
2. Register on http://launchpad.net/, which is the landing space for OpenStack bugs.
3. Sign the OpenStack CLA ( http://wiki.openstack.org/CLA) and register as a contributor.

4. Once you join the OpenStack-CLA group, you will be able to claim bugs for Nova, Glance, Swift, Keystone and Horizon services.
5. Go all in with your dev team to fix the bug, keeping in mind the documentation given in the section above. Bugs should be reproducible in order to even get started on fixing them.
6. Propose fixes and commit changes using git.

## Conclusion

As adoption of the cloud continues to accelerate, businesses will need to determine how the cloud can help them. There are many avenues to take when deploying a cloud as we have seen in this document and many technologies to choose from. OpenStack is one of many frameworks available to help lay the foundation to your own cloud. The open-source community continues to grow, and we see more support being provided to projects from large technology companies.

Open-source software will remain a vital component to any cloud environment and will be deployed to varying degrees. There are many materials and groups available to get one started. Finding the right use-case and technology partners will be critical when beginning your open-source cloud journey.

## References


1. Mell, P., & Grance, T. (2011, September). *The NIST definition of cloud computing*. The NIST Definition of Cloud Computing. Retrieved January 16, 2023, from https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-145.pdf
2. The World #RunsOnOpenStack. (n.d.). Retrieved January 16, 2023, from https://www.openstack.org/use-cases