Solution Architecture

To understand the possibilities Xloud OpenStack offers, it's best to start with basic architectures that are tried-and-true and have been tested in production environments. We offer example with XOS and the networking architectures; you should find the considerations made for the choices in each as well as a rationale for why it worked well in a given environment.

Because OpenStack is highly configurable, with many different backends and network configuration options, this solution document defines example architecture to simplify the task of documenting, as well as to provide the scope for this guide. The offered architecture example is currently running in production and serving users.

Overview

The architecture that can be built upon for Compute has three controller and multiple compute nodes. The simplest architecture for storage has five nodes: Two for identifying users and proxying requests to the API and for storage itself to provide enough replication for eventual consistency. This example architecture does not dictate a particular number of nodes but shows the thinking and considerations that went into choosing this architecture including the features offered.

Components

Component	Details
OpenStack release	Zed
Host operating system	XOS
OpenStack package repository	openstack/kolla-ansible
Hypervisor	KVM
Database	SQL / Mariadb
Message queue	RabbitMQ
Networking service	nova-network
Network manager	Neutron
Single nova-network or multi-host?	multi-host
Image Service (glance) backend	file
Identity Service (keystone) driver	SQL / Mariadb
Block Storage Service (cinder) backend	Ceph
Live Migration backend	Shared storage using Ceph
Object storage	OpenStack Object Storage (swift)

Note

The following features of OpenStack are supported by the example architecture documented in this guide, but are optional:

- Dashboard: You probably want to offer a dashboard, but your users may be more interested in API access only.
- Block storage: You don't have to offer users block storage if their use case only needs ephemeral storage on compute nodes, for example.
- Floating IP address: Floating IP addresses are public IP addresses that you allocate from a predefined pool to assign to virtual machines at launch. Floating IP address ensure that the public IP address is available whenever an instance is booted. Not every organization can offer thousands of public floating IP addresses for thousands of instances, so this feature is considered optional.
- Live migration: If you need to move running virtual machine instances from one host to another with little or no service interruption, you would enable live migration, but it is considered optional.
- Object storage: You may choose to store machine images on a file system rather than in object storage if you do not have the extra hardware for the required replication and redundancy that OpenStack Object Storage offers.

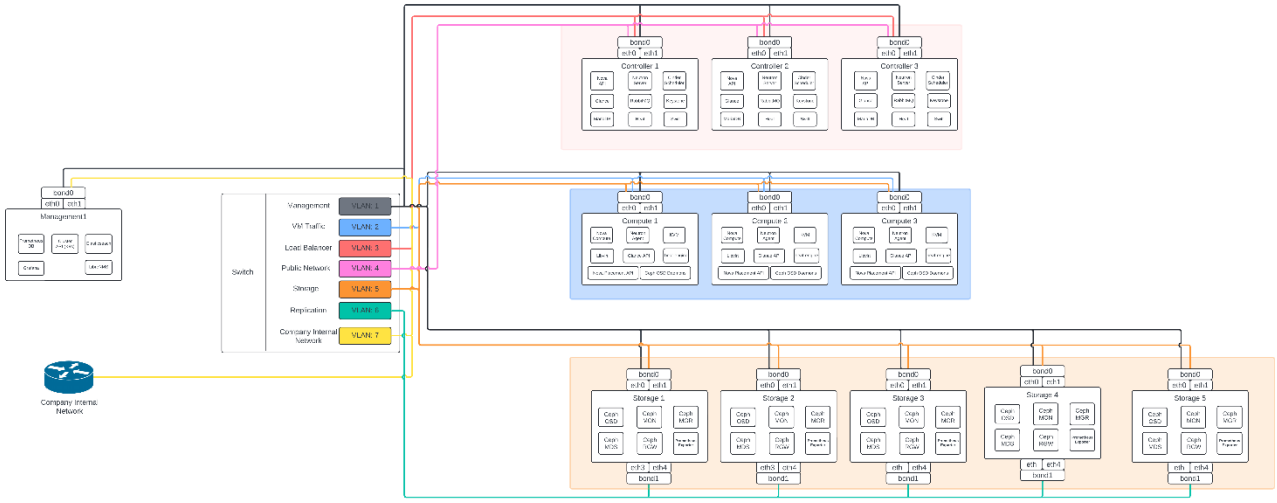

Figure 1.1, Deployment Architecture

Scan QR for High-Res Diagram

Rationale

This example architecture has been selected based on the current default feature set of OpenStack Zed, with an emphasis on stability. Many clouds currently run OpenStack in production have made similar choices.

First choosing the operating system that runs on all of the physical nodes. While OpenStack is supported on several distributions of Linux, we used XOS which is used for the development, has feature completeness compared with other distributions and has clear future support plans.

We recommend that you do not use the default XOS OpenStack install packages and instead use the openstack kolla ansible. The Openstack kolla ansible is a package repository supported by openstack that allows you to upgrade to future OpenStack releases.

KVM as a hypervisor complements the choice of XOS—being a matched pair in terms of support, and also because of the significant degree of attention it garners from the OpenStack development community (including the authors, who mostly use KVM). It is also feature complete, free from licensing charges and restrictions.

MariaDB follows a similar trend. Despite its divergence from MySQL after its acquisition, MariaDB has become a widely trusted and thoroughly tested database for OpenStack deployments. It maintains compatibility with MySQL and has a strong, active community that ensures continued innovation and performance improvements. MariaDB offers more open-source-driven development and is a highly documented database system, making it ideal for production environments. We deviate from the default database, SQLite, as SQLite is not suited for production-level OpenStack usage. MariaDB's robustness and scalability make it a preferred choice for OpenStack deployments.

The choice of *RabbitMQ* over other AMQP compatible options that are gaining support in OpenStack, such as ZeroMQ and Qpid, is due to its ease of use and significant testing in production. It also is the only option that supports features such as Compute cells. We recommend clustering with RabbitMQ, as it is an integral component of the system and fairly simple to implement due to its inbuilt nature.

There are several options for networking in OpenStack Compute. We recommend *neutron* and to use *multi-host* networking mode for high availability, running one nova-network daemon per OpenStack compute host. This provides a robust mechanism for ensuring network interruptions are isolated to individual compute hosts and allows for the direct use of hardware network gateways.

Live Migration is supported by way of shared storage, with *Ceph* as the distributed file system.

Acknowledging that many small-scale deployments see running Object Storage just for the storage of virtual machine images as too costly, we opted for the file backend in the OpenStack Image Service (Glance). If your cloud will include Object Storage, you can easily add it as a backend.

We chose the MariaDB *backend for Identity Service (keystone)* over others, such as LDAP. This backend is simple to install and is robust.

Block Storage (cinder) is installed natively on external storage nodes and uses the Ceph/*LVM/iSCSI plug-in*. Most Block Storage Service plug-ins are tied to particular vendor products and implementations limiting their use to consumers of those hardware platforms, but Ceph/LVM/iSCSI is robust and stable on commodity hardware.

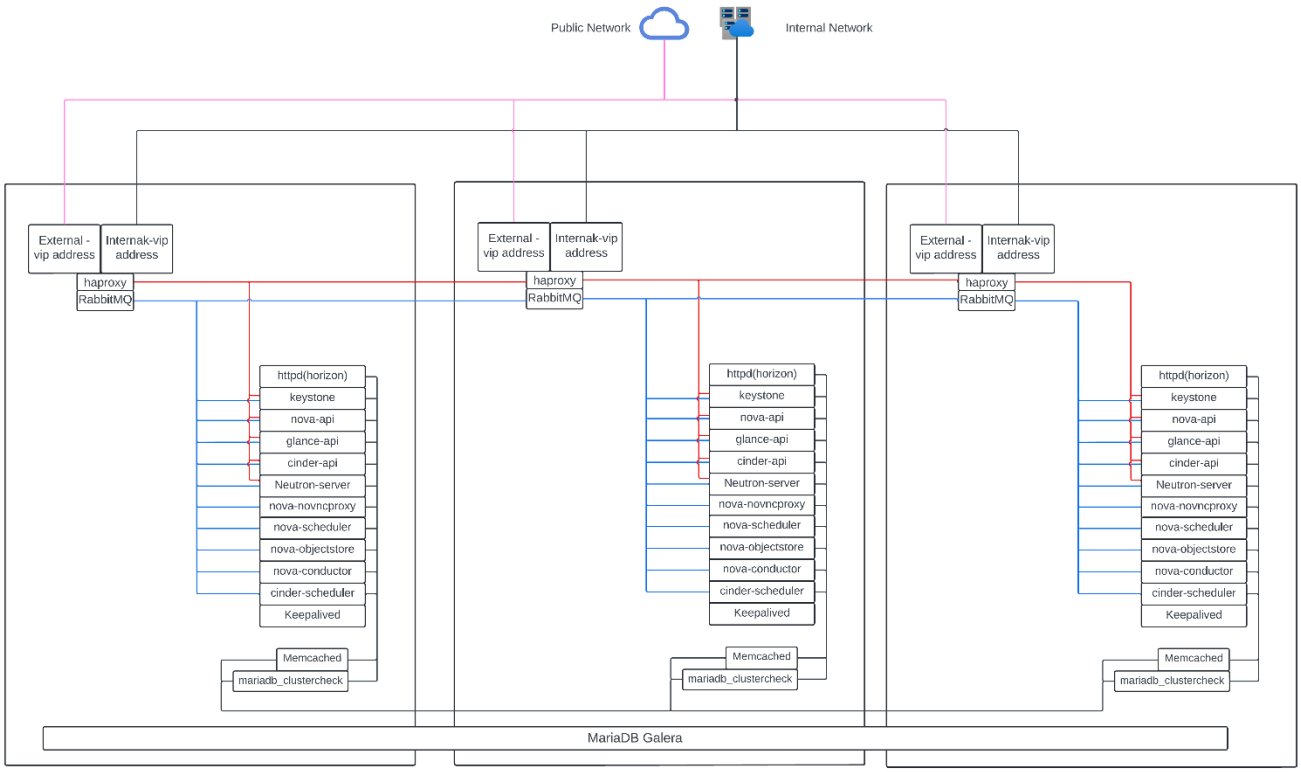

Figure 1.2, Controller nodes

Figure 1.2

Scan QR for High-Res Diagram

Figure 1.3

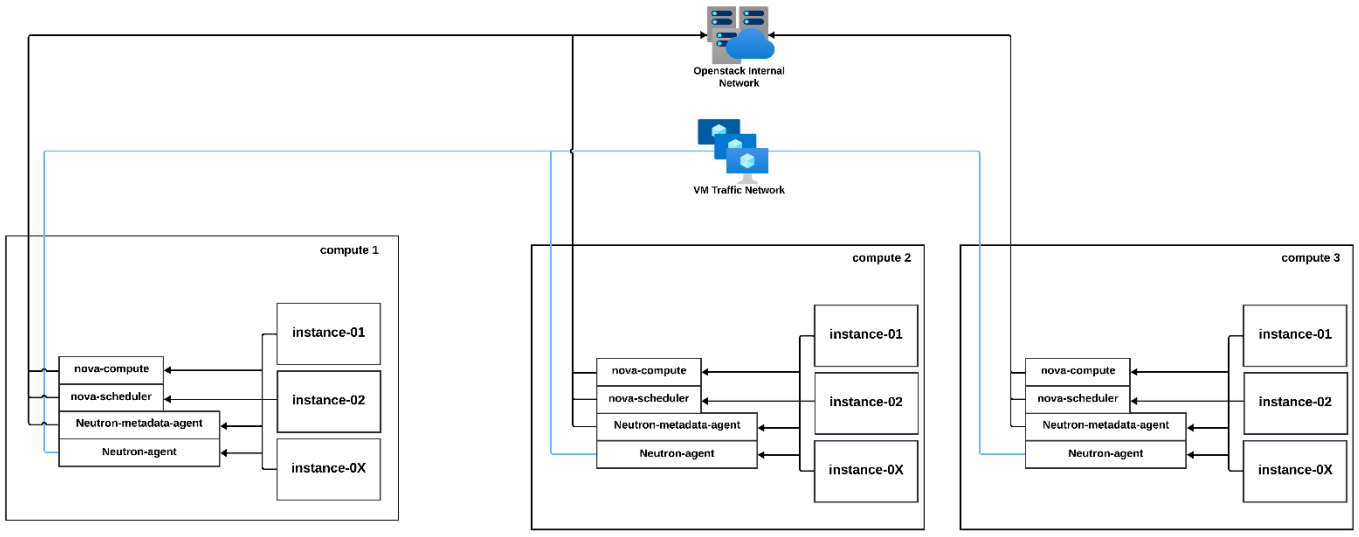

Figure 1.3, Compute nodes

Node types

This section gives you a breakdown of the different nodes that make up the OpenStack environment. A node is a physical machine that is provisioned with an operating system and running a defined software stack on top of it. Table 1.1, “Node types” provides node descriptions and specifications.

Type	Description
Controller	Controller nodes are responsible for running the management software services needed for the OpenStack environment to function. These nodes: • Provide the front door that people access as well as the API services that all other components in the environment talk to. • Run a number of services in a highly available fashion, utilizing Pacemaker and HAProxy to provide a virtual IP and load-balancing functions so all controller nodes are being used. • Supply highly available "infrastructure" services, such as MariaDB and RabbitMQ, that underpin all the services. • Provide what is known as "persistent storage" through services run on the host as well. This persistent storage is backed onto the storage nodes for reliability. See Figure 1.2, “Controller node”.
Compute	Compute nodes run the virtual machine instances in OpenStack. They: • Run the bare minimum of services needed to facilitate these instances. • Use local storage on the node for the virtual machines so that no VM migration or instance recovery at node failure is possible. See Figure 1.3, “Compute node”.
Storage	Storage nodes store all the data required for the environment, including disk images in the Image Service library, and the persistent storage volumes created by the Block Storage service. Storage nodes use Ceph technology to keep the data highly available and scalable.
Network	Network nodes are responsible for doing all the virtual networking needed for people to create public or private networks and uplink their virtual machines into external networks. Network nodes: • Form the only ingress and egress point for instances running on top of OpenStack. • Run all the environment's networking services, with the exception of the networking API service (which runs on the controller node).
Management	Management nodes are used by internal administration staff only to provide a number of basic system administration functions needed to get the environment up and running and to maintain the hardware, OS, and software on which it runs. These nodes run services such as provisioning, configuration management, monitoring and management software. They are not required to scale, although these machines are usually backed up.

Networking layout

The network contains all the management devices for all hardware in the environment (for example, by including Dell iDrac devices for the hardware nodes, and management interfaces for network switches). The network is accessed by internal staff only when diagnosing or recovering a hardware issue.

OpenStack Internal/Management Network

This network is used for OpenStack management functions and traffic, including services needed for the provisioning of physical nodes (pxe, tftp, kickstart), traffic between various OpenStack node types using OpenStack APIs and messages (for example, nova-compute talking to keystone or cinder-volume talking to nova-api), and all traffic for storage data to the storage layer underneath by Ceph. All physical nodes have at least one network interface (typically eth0) in this network. This network is only accessible from other VLANs on port 22 (for ssh access to manage machines).

VM traffic network

This is a closed network that is not publicly routable and is simply used as a private, internal network for traffic between virtual machines in OpenStack, and between the virtual machines and the network nodes that provide l3 routes out to the public network (and floating IPs for connections back into the VMs). Because this is a closed network, we are using a different address space to the others to clearly define the separation. Only Compute and OpenStack Networking nodes need to be connected to this network.

Load Balancer Network

This network is used by the controller nodes for distributing network traffic across multiple services or resources to ensure high availability and balance the load. It is responsible for managing traffic to critical services like APIs, ensuring efficient distribution of requests. The network handles traffic between clients (both external and internal) and OpenStack services, providing redundancy.

Public Network

This network is a combination of:

- IP addresses for public-facing interfaces on the controller nodes (which end users will access the OpenStack services)
- A range of publicly routable, IPv4 network addresses to be used by OpenStack Networking for floating IPs. You may be restricted in your access to IPv4 addresses; a large range of IPv4 addresses is not necessary.
- Routers for private networks created within OpenStack.

This network is connected to the controller nodes so users can access the OpenStack interfaces and connected to the network nodes to provide VMs with publicly routable traffic functionality. The network is also connected to the utility machines so that any utility services that need to be made public (such as system monitoring) can be accessed.

Storage Network

This network is used for communication between Ceph storage nodes and compute nodes. It handles the storage of VM volumes and data by enabling communication between the compute nodes and the Ceph cluster for read/write operations. This dedicated storage network ensures that data traffic between VMs and storage systems does not interfere with management or other network traffic.

Replication Network

This network is exclusively used by Ceph storage nodes to replicate data between each other. It handles Ceph's replication traffic, ensuring data redundancy and fault tolerance by synchronizing data across multiple storage nodes. This separation of replication traffic ensures high availability and performance by preventing it from impacting storage or management traffic.

Company Internal Network

This network is used for accessing management nodes by company staff or administrators. It is typically reserved for internal operations, monitoring, and administrative access. It enables secure, internal communication for managing and accessing the OpenStack environment, isolated from external public traffic. Access is typically restricted to internal resources.

Node connectivity

The following section details how the nodes are connected to the different networks and what other considerations need to take place (for example, bonding) when connecting nodes to the networks.

Initial deployment

Initially, the connection setup should revolve around keeping the connectivity simple and straightforward to minimize deployment complexity and time to deploy. The deployment shown in aims to have 2 x 10G connectivity available to all compute nodes, while still leveraging bonding on appropriate nodes for maximum performance as well as providing more network bandwidth to the storage layer.

Component	Node type	Availability	Scalability
Dashboard (horizon)	Controller	The dashboard is run on all controller nodes, ensuring at least one instance will be available in case of node failure. It also sits behind HAProxy, which detects when the software fails and routes requests around the failing instance.	The dashboard is run on all controller nodes, so scalability can be achieved with additional controller nodes. HAProxy allows scalability for the dashboard as more nodes are added.
Identity (keystone)	Controller	Identity is run on all controller nodes, ensuring at least one instance will be available in case of node failure. Identity also sits behind HAProxy, which detects when the software fails and routes requests around the failing instance.	Identity is run on all controller nodes, so scalability can be achieved with additional controller nodes. HAProxy allows scalability for Identity as more nodes are added.
Image Service (glance)	Controller	The Image Service is run on all controller nodes, ensuring at least one instance will be available in case of node failure. It also sits behind HAProxy, which detects when the software fails and routes requests around the failing instance.	The Image Service is run on all controller nodes, so scalability can be achieved with additional controller nodes. HAProxy allows scalability for the Image Service as more nodes are added.
Compute (nova)	Controller, Compute	The nova API, scheduler, objectstore, novncproxy and conductor services are run on all controller nodes, ensuring at least one instance will be available in case of node failure. Compute is also behind HAProxy, which detects when the software fails and routes requests around the failing instance.	The nova API, scheduler, objectstore, novncproxy and conductor services are run on all controller nodes, so scalability can be achieved with additional controller nodes. HAProxy allows scalability for Compute as more nodes are added. The scalability of services running on the

Component	Node type	Availability	Scalability
		Compute's compute and conductor services, which run on the compute nodes, are only needed to run services on that node, so availability of those services is coupled tightly to the nodes that are available. As long as a compute node is up, it will have the needed services running on top of it.	compute nodes (compute, conductor) is achieved linearly by adding in more compute nodes.
Block Storage (cinder)	Controller	Block Storage API, scheduler, and volume services are run on all controller nodes, ensuring at least one instance will be available in case of node failure. Block Storage also sits behind HAProxy, which detects if the software fails and routes requests around the failing instance.	Block Storage API, scheduler and volume services are run on all controller nodes, so scalability can be achieved with additional controller nodes. HAProxy allows scalability for Block Storage as more nodes are added.
OpenStack Networking (neutron)	Controller, Compute, Network	The OpenStack Networking service is run on all controller nodes, ensuring at least one instance will be available in case of node failure. It also sits behind HAProxy, which detects if the software fails and routes requests around the failing instance. OpenStack Networking's ovs/ovn-agent, l3-agent-dhcp-agent, and metadata-agent services run on the network nodes, as lsb resources inside of Pacemaker. This means that in the case of network node failure, services are kept running on another node. Finally, the ovs/ovn-agent service is also run on all compute nodes, and in case of compute node failure, the other nodes will continue to function using the copy of the service running on them.	The OpenStack Networking server service is run on all controller nodes, so scalability can be achieved with additional controller nodes. HAProxy allows scalability for OpenStack Networking as more nodes are added. Scalability of services running on the network nodes is not currently supported by OpenStack Networking, so they are not considered. One copy of the services should be sufficient to handle the workload. Scalability of the ovs/ovn-agent running on compute nodes is achieved by adding in more compute nodes as necessary.