# Conceptual architecture

The architecture includes 4 parts, there are Controller, Compute, Network and Share Storage.

1. Controller

The Controller node runs the Identity Service, Image Service, dashboard, Block Storage Service, Meter Service and management portion of Compute. It also contains the related API Services, MySQL databases and messaging system.

1. Compute

The Compute node runs the hypervisor portion of Compute, which operates tenant virtual machines. By default, Compute uses KVM as the hypervisor. Compute also provisions and operates tenant networks and implements security groups.

1. Network

The Network node runs the tenant network and the vm instance connectivity, with both other vms and external network. Network also provisions associated network services, such as NAT, Firewall, Load Balance, QoS and so on.

1. Share Storage

The Share Storage runs the storage service for such as compute, block storage, image and which needs storage. It can provisions Object, File System and Block storage. In this architecture, it is the storage backend for compute Root and Ephemeral Storage, Volume Storage and OS Image Storage.

# Example architecture

There are 3 nodes for Controller, 4 nodes with 6\*4T disks for Compute and Share Storage, 2 nodes for Network & Monitor.

* 1. Controller

The Controller Cluster contains 3 nodes and runs OpenStack API & Schedulers, MySQL Cluster, RabbitMQ Cluster and Memcached Cluster.

* 1. Compute

The Compute Cluster contains 4 nodes and runs Nova Compute, Neutron L2 Agent, Neutron L3 Agent, DHCP Agent, Ceph OSD.

* 1. Network & Monitor

The Network Cluster contains 2 nodes and runs Neutron Server, HAproxy, Ceph Monitor, Pacemaker Cluster Stack, Nagios.

# High Availability

High Availability systems seek to minimize two things:

* System downtime
* Data losssss

In OpenStack, we can implements HA through service state type and configuration mode.

## Stateless vs. Stateful services

A stateless service is one that provides a response after request, and then requires no further attention(don’t check if the service is actually running). To make a stateless service highly available, need to provide redundant instances and load balance them. OpenStack services that are stateless include nova-api, nova-conductor, glance-api, keystone-api, neutron-api and nova-scheduler.

A stateful service is one where subsequent requests to the service depend on the results of the first request. Simply providing additional instances and load balancing will not solve the problem. OpenStack services that are stateful include the OpenStack database and message queue.

Making highly available can depend on the active/passive or active/active configuration. In order to simplify the system, we will mostly adopt active/active configuration.

## HA Using Active/Active

### MySQL with Galera

MySQL/Galera is synchronous multi-master cluster for MySQL/InnoDB database

### RabbitMQ

RabbitMQ is the default AMQP server used by many OpenStack services. Making the RabbitMQ service highly available involves:

* Install RabbitMQ
* Configure RabbitMQ for HA queues
* Configure OpenStack services to use Rabbit HA queues

### HAproxy Nodes

HAProxy is a very fast and reliable solution offering high availability, load balancing, and proxying for TCP and HTTP-based applications.

Also we have to consider that this service should not be a single point of failure, so we need two nodes running HAproxy at least.

### OpenStack Controller Nodes

OpenStack Controller Nodes contains:

* All OpenStack API services
* All OpenStack schedulers
* Memcached service

#### OpenStack API & schedulers

##### API Services

All OpenStack projects have an API service for controlling all the resources in the Cloud. In Active/Active mode, the most common setup is setup redundant services and use load-balancing and virtual IP (with HAproxy & Keepalived).

To configure HA API services, we need to:

* Using Virtual IP when configuring OpenStack Identity Endpoints.
* All OpenStack configuration files should refer to Virtual IP.

Because in Active/Active, we don’t check if the service is actually running, so all OpenStack API should be monitored by another tool (i.e. Nagios) to detect failures.

##### Schedulers

OpenStack schedulers are used to determine how to dispatch compute, network and volume requests. The most common setup is to use RabbitMQ. Those services are connected to the messaging backend and can scale-out :

* nova-scheduler
* nova-conductor
* cinder-scheduler
* neutron-server
* ceilometer-collector

#### Memcached

Most OpenStack services use memcached to offer persistence and store ephemeral datas (like keystone tokens). Memcached can scale-out easily without specific trick.

### OpenStack Network Nodes

OpenStack Network Nodes contains:

* Neutron DHCP Agent
* Neutron L2 Agent
* Neutron L3 Agent
* Neutron Metadata Agent
* Neutron LBaaS Agent

**Note**: The Neutron L2 Agent does not need to be highly available. One L2 agent runs per node and controls its virtual interfaces. That’s why it cannot be distributed and highly available.

#### Neutron DHCP Agent

Neutron DHCP agent distributes IP addresses to the VMs with dnsmasq (by default).

Since the Grizzly release, OpenStack Networking service has a scheduler that lets you run multiple agents across nodes. Also, the DHCP agent can be natively highly available.

#### Neutron L3 Agent

The Neutron L3 agent provides L3/NAT forwarding to ensure external network access for VMs on tenant networks.

Since the Grizzly release, the Neutron L3 Agent is scalable natively. But there is no native feature to make these routers highly available. At this time, the Active/Passive solution exists to run the Neutron L3 agent in failover mode with Pacemaker.

#### Neutron Metadata Agent

Neutron Metadata agent allows Nova API Metadata to be reachable by VMs on tenant networks.

There is no native feature to make this service highly available. At this time, the Active / Passive solution exists to run the Neutron Metadata agent in failover mode with Pacemaker.

## HA Using Active/Passive

### The Pacemaker Cluster Stack

OpenStack HA relies on the [Pacemaker](http://www.clusterlabs.org/) cluster stack, the state-of-the-art high availability and load balancing stack for the Linux platform.

### OpenStack Network Nodes

#### Neutron L3 Agent

High Availability for the L3 agent is achieved by adopting Pacemaker.

#### Neutron Metadata Agent

High Availability for the Metadata agent is achieved by adopting Pacemaker.

## Network HA

### Agents

Previous versions of OpenStack Network (formerly known as Neutron) had single point of failure where the agents were only deployable on a single network node. We can solve this problem by deploying the L2, L3 and DHCP networking agents to each compute node. This has the advantage of enabling the Neutron scheduler to schedule a new node to act as the L2, L3 and DHCP agents for the tenant network in the event of a failure. The following agents are run on each Compute node.

* L2 agent.
* L3 Agent.
* DHCP Agent.

These agents provide L2, L3 and DHCP services for the tenant networks. When a new tenant network is created, the Neutron scheduler select which node and its L2, L3 and DHCP agents will be used to supply networking services to the tenant network. In the event of a failed node, the tenant networking will be rescheduled on one of the remaining available node in the Compute cluster. At a tenant networking level this is still a single point of failure, but is somewhat mitigated by the fact that the tenant networking and VM can be rescheduled on one or the remaining node in the Compute cluster.

### Scheduler

The Neutron scheduler components are run on three or more controller nodes to increase the overall resilience of the component cluster. This scheduler-cluster will determine which of the compute node L2, L3 and DHCP agents will be responsible for creating and running the tenant network. The Neutron server components are placed behind the system load-balancer to balance incoming HTTP requests. In the event of a failed node, the load-balancer will remove the node from its list of nodes available to handle requests and incoming HTTP requests will be distributed between the remaining nodes.