Mirantis OpenStack Planning Guide

version 9.2

Contents

Copyright notice

Preface

This documentation provides information on how to use Fuel to deploy OpenStack environments. The information is for reference purposes and is subject to change.

Intended Audience

This documentation is intended for OpenStack administrators and developers; it assumes that you have experience with network and cloud concepts.

Documentation History

The following table lists the released revisions of this documentation:

Revision Date	Description
February 6, 2017	9.2 GA

Introduction

OpenStack is an extensible, versatile, and flexible cloud management platform. It is a portfolio of cloud infrastructure services — compute, storage, networking, and other core resources that are exposed through REST APIs. It enables a wide range of control over these services, both from the perspective of an integrated Infrastructure as a Service (IaaS) controlled by applications and as a set of tools that enable automated manipulation of the infrastructure itself.

This document will help cloud administrators to understand the OpenStack components and architecture they can deploy using Fuel, as well as estimate future scaling requirements. This document presumes that you are familiar with basic OpenStack concepts.

This section includes the following topics:

OpenStack environment architecture

Using Fuel, you can deploy multiple highly-available OpenStack environments. An OpenStack environment consists of physical servers, or node servers, of the following types:

Controller

Controller nodes manage all operations within an OpenStack environment. To achieve high availability, deploy at least three controller nodes for quorum. However, if you do not have enough hardware to deploy three controllers, you can start with one controller node and add more nodes later. Similarly, if you later need to expand your controller cluster, you add more controller nodes.

Compute

Compute nodes provide processing resources to accommodate virtual machine workloads. The virtual machines, or instances, that you create in Horizon, run on the compute nodes. Such components as neutron-agent and ceilometer-agent-compute may also run on the compute nodes. The nova-compute service controls the lifecycle of virtual machines and typically runs on the compute nodes. However, if you deploy an OpenStack environment with VMware vCenter as a hypervisor, nova-compute runs on the controller nodes.

Storage

The storage needs of your environment can be addressed either by provisioning dedicated storage nodes or by assigning a storage role to the controller node. Fuel deploys the following storage options:

- **Cinder Logical Volume Manager (LVM)**

  Provides persistent block storage to virtual machines over iSCSI or iSCSI over RDMA. If you configure a dedicated storage node, the cinder-volume service runs on the storage node.

- **Ceph**

  Combines object and block storage. You can use Ceph RADOS Block Device (RBD) as storage for Ceph volumes, Glance images, and/or ephemeral volumes. You can also deploy Ceph RadosGW for objects. If you create a separate Ceph storage node, it runs the Ceph object storage daemon (OSD).

> Warning
>
> We highly recommend that you do not place Ceph OSD on controllers as it severely degrades the controller's performance. Use separate storage nodes if you have enough hardware.

- **Swift**

  Provides object storage that Glance can use to store virtual machine images and snapshots. If you do not select any other storage options when you provision an OpenStack environment, Fuel deploys Swift by default. Applications that you deployed in your OpenStack environment may also use Swift object storage.

Selection of a storage back end for your OpenStack environment depends on multiple factors. For more information, see: Plan the Storage.

> Seealso
>
> - Fuel reference architecture overview

Introduction to Fuel

Fuel is a deployment automation tool with a user-friendly web UI that helps you to quickly provision an OpenStack environment. Fuel is a part of Mirantis OpenStack — a productized snapshot of the open source technologies. Fuel includes scripts that dramatically facilitate and speed up the process of cloud deployment, without requiring you to completely familiarize yourself with the intricate processes required to install the OpenStack environment components.

Fuel includes the following components:

Fuel Master node

A standalone physical or virtual machine server from which you provision your OpenStack environments.

Fuel Slave nodes

Servers, or node servers, that can be controller, compute, storage, or other nodes.

Fuel deploys an OpenStack architecture that was thoroughly tested and proved to be effective on many deployments. One of the main advantages of the architecture is high availability and load balancing. All components that Fuel deploys on the controller node are highly-available. For example, to achieve high availability in stateless OpenStack services, such as nova-api, nova-conductor, and so on, Fuel uses Pacemaker/Corosync <http://clusterlabs.org/>. For more information, see: Fuel reference architecture overview.

> Note
>
> If the architecture that Fuel deploys does not meet your business requirements, standard configuration can be adjusted according to your needs. For more information, contact Managed Services for Mirantis OpenStack

Overview of the planning process

The following table will help you to plan your deployment correctly:

Step description	Additional information
Review system requirements and verify that your environment meets the requirements.	See Fuel system requirements
Review Fuel reference architecture.	See: Fuel reference architecture overview
Select a network topology.	See: Plan the network
Prepare an IP address management plan and network association.	Identify the network addresses and VLAN IDs for your Public, Admin (PXE), Management, and Storage networks. Prepare a logical network diagram.
Select storage.	See: Plan the storage
Determine how many nodes to deploy and which roles to assign to each and the high availability to implement.	See: Nodes and roles
Plan monitoring facilities	See: Plan the monitoring tools
If you want to run Hadoop, plan to install Sahara.	See: Plan a Hadoop cluster
If you want to use VMware vSphere as a hypervisor, plan the vSphere integration.	See: Plan the vSphere integration
Calculate the server and network hardware needed.	See: Calculate hardware resources

Fuel system requirements

This section provides system requirements for the Fuel Master node and recommendations for Fuel Slave nodes. Fuel Slave nodes are the OpenStack and other nodes. Therefore, hardware configuration that you will have to use for Fuel Slave nodes will vary depending on the type of cloud that you build.

This section includes the following topics:

Network requirements

The server on which you install the Fuel Master node must have access to the Internet to download packages required to install the operating system on the Fuel Slave nodes and later update the packages, as well as the Fuel software. If for security or other reasons the Fuel Master node does not have an Internet connection, then you must configure a local repository mirror and point Fuel to use this repository.

All nodes in your OpenStack environment must be able to communicate with each other. Your network configuration greatly depends on your environment requirements and the network topology that you select. For more information, see: Plan the network.

This section includes the following topics:

Public and floating IP address requirements

This section describes the OpenStack requirements for public and floating IP addresses. The requirements differ depending on the selected network topology.

The following table provides the floating and public IP address requirements for Neutron:

Public and floating IP requirements for Neutron:

IP range	Description
Public IP range	Required: • 1 IP address per controller node. This IP address is assigned to the external network (br-ex). • 2 IP addresses for environment's virtual IP addresses • 1 IP address for the default gateway Optional: • If you use Neutron DVR, it requires one additional IP address for each Compute node in case you plan to use Floating IPs in the deployment. • If you deployed Zabbix, 1 IP address per Zabbix node.

Floating IP range	Required: • 1 IP address per tenant, including the Admin tenant. This IP addresses are assigned to the virtual machine interfaces through the tenant's virtual router. Therefore, one Floating IP is assigned to the Admin tenant automatically as part of the OpenStack deployment process. • 1 IP address per virtual machine connected to the external network. These IP addresses are assigned on demand and may be released from the virtual machines and returned back to the pool of unassigned Floating IP addresses.

Seealso

- Calculate floating and public IP addresses
- OpenStack Networking Guide

Routing requirements

If you deploy a single node group, you do not need to meet any routing requirements. However, if you deploy multiple node groups, your environment must meet the following requirements:

- Node groups must communicate with each other through routers.
- Each network in each node group must have a gateway.

When you configure routing for your OpenStack network, consider the following recommendations:

- Use the default routing through a router in the Public network
- Use the management network to access your management infrastructure, including L3 connectivity, if required.
- Isolate Storage and Private networks from all other L3 networks.

If you plan to use IPv6 addresses, verify that your environment meets these additional routing requirements:

- The provider's network router supports IPv6 routing.
- The tenant's network router is connected to the Fuel Public network and provides the functionality to configure static routes to the virtual routers in the OpenStack environment.

Switching requirements

Before deploying an OpenStack environment, you must configure network switches. Since terminology and configuration vary for each vendor, this section provides general vendor-agnostic information about how traffic should flow. For detailed information on how to set up networks, read the OpenStack Networking Guide.

Disregarding the type of the switch and vendor that you use, set the following configuration for Admin (PXE) network:

1. Configure all access ports to allow non-tagged PXE booting connections from each Fuel Slave node to the Fuel Master node.
2. Set the switch port assigned for PXE requests on all nodes to Access mode.
3. Isolate the PXE boot network from any other L2 network segments. Since Fuel runs its own DHCP server, it may conflict with DHCP servers from other L2 network segments which may result in both company infrastructure and Fuel being nonoperational.
4. Configure each of the switch ports connected to the Fuel Slave nodes as STP Edge port or spanning-tree port fast trunk to avoid DHCP timeouts. For different switch port vendors, this terminology may vary.

Fuel Master node hardware requirements

When planning hardware for the Fuel Master node, verify that your hardware meets the following minimum requirements:

For a production environment:

- Quad-core CPU
- 4 GB RAM
- 10 Gigabit network port
- IPMI access through an independent management network
- 50 GB (depends on the number of deployed nodes)

  See: Storage on the Fuel Master node

For a testing environment:

- Dual-core CPU
- 2 GB RAM
- 1 Gigabit network port
- 50 GB disk
- Physical console access

Storage on the Fuel Master node

In a default OpenStack environment deployed by Fuel, compute nodes send logs to the Fuel Master node. Therefore, you must assign a sufficient amount of disk space to store these logs. During installation, Fuel automatically creates a separate /var partition to store logging information and allocates 40% of disk size to it. Depending on the size of your cloud, the log rotation interval, and the detail of logging, you may need different disk sizes. For example, in a production environment that includes 200 compute nodes, with the log rotation interval of one month, allocate 2 - 4 TB of disk space for logging.

A typical production OpenStack environment includes a monitoring plane, such as StackLight, and logging is configured to use the log aggregator of the monitoring plane. Therefore, the Fuel

Master node does not require additional disk space to store logs. However, you have to assign a required amount of disk space for logs on the monitoring nodes.

Seealso

- Monitoring Guide

Fuel Slave nodes hardware recommendations

Determining the appropriate hardware for the Slave nodes depends on the node type, the workloads that you plan to run on the nodes, and whether you combine different OpenStack components on one node or run them separately. Typically, you need a two-socket server with the CPU, memory, and disk space that meet your project requirements. Read the OpenStack Architecture Design Guide for recommendations on how to plan an OpenStack deployment.

General guidelines for the Fuel Slave nodes:

- **Controller nodes**

  Use at least three controller nodes for high availability. High availability is recommended for all production environments. However, you can start with a single controller node for testing purposes and add more nodes later. The controller nodes must form a quorum. Therefore, for all deployments, the total number of controller nodes must be odd. Further resource scaling depends on your use case and requires extensive assessment of your environment and business needs.

- **Compute nodes**

  The number and hardware configuration of the compute nodes depend on the following:

  - Number of virtual machines
  - Applications that you plan to run on these virtual machines
  - Types of workloads

- **Storage nodes**

  The number and capacity of the storage nodes highly depend on the type of the storage, redundancy, and workloads that you run on the compute nodes. Therefore, the storage configuration for every deployment will vary significantly.

Seealso

- OpenStack Architecture Design Guide
- OpenStack Operations Guide

Fuel reference architecture overview

This section provides information about the OpenStack environments that Fuel enables you to deploy. Since OpenStack is a highly adaptable platform, you can choose virtually any component including databases, Linux distributions, OpenStack components, and so on. However, Fuel does not allow you to deploy all of them. Instead, Fuel enables you to deploy best of breed, versatile, and robust OpenStack reference architecture that will address the requirements of various types of clouds.

In highly-available architecture that Fuel deploys, all components including controller nodes, databases, message queues, and so on, have redundant instances that ensure reliability and continuous availability of user data and applications. For in-depth explanation of high availability in OpenStack, read the OpenStack High Availability Guide available at http://docs.openstack.org/.

This section includes the following topics:

Nodes and roles

When planning your OpenStack deployment, you must determine a proper mix of node types and roles for each node. You assign roles to each node server when you deploy an OpenStack environment.

The planning of nodes and roles involves determining the level of HA you want to implement, as well as hardware planning.

Some general guiding principles:

If you deploy a production-grade OpenStack environment, spread the roles and the workload over as many servers as possible to have a fully redundant, highly-available OpenStack environment and to avoid performance bottlenecks.

For testing purposes, you can deploy your OpenStack environment on VirtualBox. For more information, see Fuel QuickStart Guide.

OpenStack can be deployed on smaller hardware configurations by combining multiple roles on the nodes and mapping multiple logical networks to a single physical NIC.

Some plugins require specific roles. Check the requirements of all plugins that you plan to use.

To maximize performance, carefully choose your hardware components and make sure that the performance features of the selected hardware are supported and enabled.

Node requirements

For a testing requirement, you need at least three controller nodes and one compute node.

In a production environment, Mirantis recommends that you separate storage nodes from controller nodes. This helps avoid resource contention, isolates failure domains, and allows to optimize hardware configurations for specific workloads.

To achieve this, you need a minimum of five nodes when using Swift and Cinder storage back ends, or seven nodes for a fully redundant Ceph storage cluster:

- Three controller nodes.
- One Cinder node or three Ceph OSD nodes.

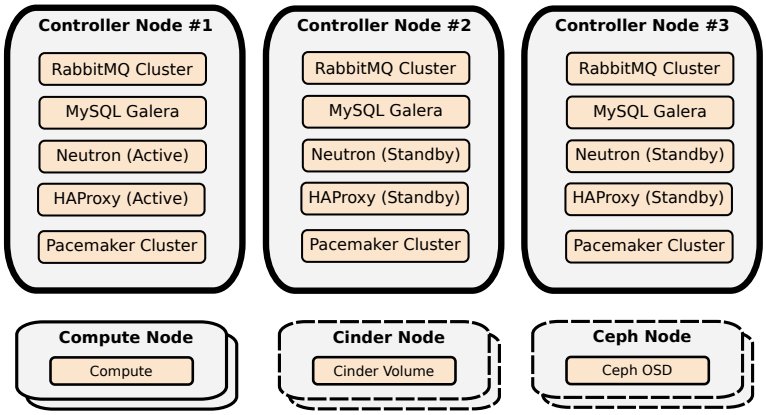
- One compute node.

> Note
>
> You do not need Cinder storage nodes if you use Ceph RBD as storage backend for Cinder volumes.

Highly-available environment architecture

A highly-available deployment must include at least three controller nodes, as well as replicas of other servers.

You can combine compute, storage, and network nodes to reduce the hardware requirements for the environment, although this may degrade the performance and robustness of the environment.

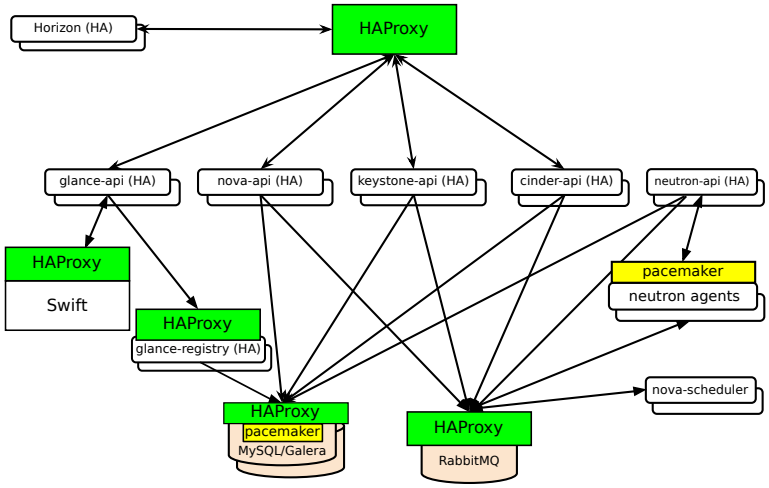


OpenStack services are interconnected by RESTful HTTP-based APIs and AMQP-based RPC messages.

Redundancy for stateless OpenStack API services is implemented through the combination of Virtual IP (VIP) management using Pacemaker and load balancing using HAProxy.

Stateful OpenStack components, such as the state database and messaging server, rely on their respective active/active and active/passive modes for HA.

For example, RabbitMQ uses built-in clustering capabilities, while the database uses MySQL/Galera replication.



Node configuration

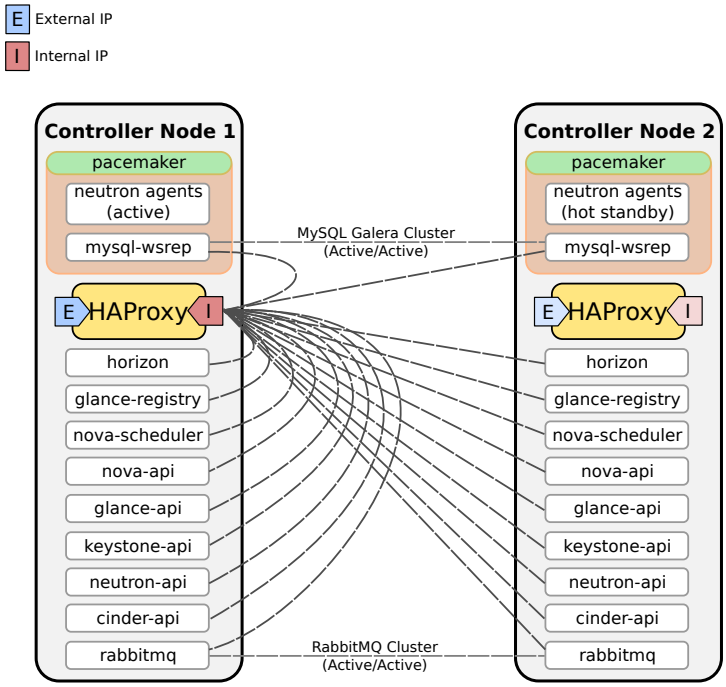
Fuel uses custom Pacemaker scripts to deploy HAProxy inside a dedicated network namespace.

Controller nodes

Controller nodes are the main components of your highly-available OpenStack environment. Controller nodes manage all operations within an OpenStack environment. Multiple controller nodes provide redundancy.

A recommended implementation of a highly available cluster architecture uses quorum-based techinques. A basic quorum-based techinque requires a simple majority of nodes: floor(N/2)+1 nodes. This means that keeping a functioning cluster with one failed controller node requires a minimum of three nodes. Keeping a functioning cluster with two failed controller nodes requires a minimum of five nodes and so on.

See also Highly-available MySQL database.



Every OpenStack controller node runs HAProxy, which performs the following:

- Manages a single External Virtual IP (VIP) for all controller nodes.
- Provides HTTP and TCP requests load balancing to the OpenStack API services, RabbitMQ, and MySQL.

> Note
>
> OpenStack services use Oslo messaging and are directly connected to the RabbitMQ nodes. Therefore, they do not require HAProxy.

When you access an OpenStack cloud through the Horizon dashboard or through the REST API:

1. Your request is sent to the controller node that holds the external VIP.
2. HAProxy terminates the connection.
3. HAProxy then handles a new request by sending the request to the original or other controller node depending on the current controller workload.

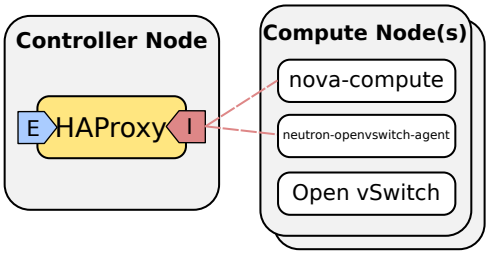
Each component or service that runs on the controller nodes has its own HA mechanism:

- Stateless OpenStack services, such as nova-api, glance-api, keystone-api, neutron-api, nova-scheduler, cinder-api, do not require any special attention besides load balancing.
- Horizon, as a typical web application, requires sticky sessions to be enabled at the load balancer.

- RabbitMQ provides active/active high availability for the OpenStack Telemetry service using mirrored queues. RabbitMQ is deployed with custom resource agent scripts for Pacemaker. You can enable HA for RPC queues as well if required. However, it may impact performance in large deployments.
- MySQL databases achieve high availability through MySQL/Galera cluster and custom resource agent scripts for Pacemaker.
- Neutron agents are active/passive and are managed by custom resource agent scripts for Pacemaker.
- Ceph monitors implement their own quorum-based HA mechanism and require time synchronization between all nodes. Clock drift higher than 50ms may break the quorum or even crash the Ceph service.

Compute nodes

Compute nodes are the servers on which your users create virtual machines (VMs) and host applications. Compute nodes communicate with the controller nodes and the essential services such as RabbitMQ and MySQL.



Storage nodes

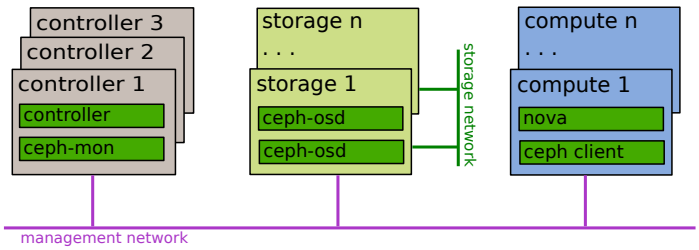
Storage nodes provide storage for persistent and temporary data. Fuel provides the following storage options:

- Cinder Logical Volume Manager (LVM)

  Cinder LVM does not implement data redundancy across nodes. If a Cinder node is degraded, volumes stored on the storage node cannot be recovered from the data stored on other Cinder nodes. If you need your block storage to be resilient, use Ceph for volumes.

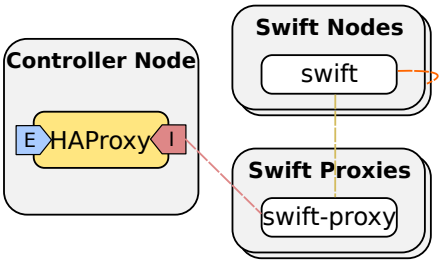
- Ceph

  Ceph implements its own HA. You must have a sufficient number of controller nodes running the Ceph Monitor service to form a quorum, and enough Ceph OSD nodes to satisfy the object replication factor.



- Swift

  Swift API relies on the same HAProxy setup with VIP on the controller nodes as the other REST APIs. In a testing environment, you can deploy Swift Storage and Proxy services on the controller nodes. In a production environment, you need dedicated nodes: two for Swift Proxy and a minimum of three for Swift Storage.



Highly-available MySQL database

OpenStack services, such as Nova, Neutron, Cinder, Glance, and Keystone require reliable databases. To ensure high availability for these services, Fuel implements the MySQL/Galera clustering technology.

MySQL with Galera implements true active/active high availability. Fuel configures MySQL/Galera to have a single active node that processes read and write operations.

MySQL/Galera includes the following features:

- The standby masters do not have the "slave lag" that is typical for MySQL master/slave topologies. This is because Galera employs synchronous replication and ensures that each cluster node is identical.
- Mirantis OpenStack uses Pacemaker and HAProxy to manage MySQL/Galera:
  - Pacemaker manages the individual MySQL/Galera nodes, HAProxy, and the Virtual IP Address (VIP).
  - HAProxy runs in the dedicated network namespace and manages connections between the MySQL/Galera active master, backup masters, and the MySQL clients connecting to the VIP.
- Only one MySQL/Galera master is active in the VIP. This single direction synchronous replication provides better performance.

Failover initiates the following actions:

1. The node that is tied to the VIP serves new data updates and increases its global transaction ID number (seqno).
2. Each other node in the Galera cluster then synchronizes its data with the node that has the seqno value greater than its current values.
3. If the status of any node falls behind the Galera cache, an entire replica is distributed to that node. This causes one of the master nodes to switch to the Donor role, allowing an out-of-sync node to catch up.

Server load balancing using HAProxy

HAProxy provides load balancing, proxying, and HA. Each OpenStack controller runs HAProxy which manages a single External Virtual IP (VIP) for all controller nodes and provides HTTP and TCP load balancing of requests that go to OpenStack API services, RabbitMQ, and MySQL.

Fuel configures HAProxy frontend for MySQL/Galera to use only one active node, while the other nodes in the cluster remain passive.

See HAProxy documentation.

Cluster resource management using Pacemaker and Corosync

Fuel implements highly-available cluster services through Corosync and Pacemaker.

At its essence, Corosync enables servers to communicate as a cluster, and Pacemaker provides the ability to control the cluster's behavior.

Corosync functions as a communication and quorum service through the Cluster Information Base (CIB) component of Pacemaker and the pcs tool. The main Cororsync configuration file is located in /etc/corosync/corosync.conf.

Corosync uses the Totem Single-Ring Ordering and Membership protocol, or the Totem protocol, to provide connectivity between the OpenStack nodes, ensure the cluster has quorum, as well as provide a data layer for services that use Virtual Synchrony.

The Totem protocol provides the following functionality:

- Ensures connectivity between cluster nodes.
- Decides if a cluster is quorate to provide services.
- Provides data layer for services that use features of Virtual Synchrony.

The following text is an example of the configuration section in the corosync.conf file:

```
totem {
  version:                       2
  token:                         3000
  token_retransmits_before_loss_const: 10
  join:                          60
  consensus:                        3600
  vsftype:                       none
  max_messages:                        20
  clear_node_high_bit:                 yes
  rrp_mode:                        none
  secauth:                       off
  threads:                       0
  interface {
    ringnumber:  0
    bindnetaddr: 10.107.0.8
    mcastaddr:   239.1.1.2
```

```
   mcastport:   5405
  }
}
```

Corosync uses UDP unicast transport and configures a "redundant ring" for communication. Fuel deploys controllers with one redundant ring. Fuel uses the default Corosync configuration. You can modify the default configuration in Fuel Puppet manifests.

Pacemaker functions in Fuel as the cluster resource manager for Neutron, HAProxy, virtual IP addresses, some OpenStack and other services. You can put your own services under Pacemaker control to leverage the use of Pacemaker as a cluster resource manager.

This is realized through Open Cluster Framework agent scripts.

The scripts performs the following:

- Starts, stops, and monitors Neutron services.
- Manages HAProxy.
- Manages virtual IP addresses.
- Manages MySQL replication.

Components of the script are located in the following directories:

- /usr/lib/ocf/resource.d/mirantis/ocf-neutron-[metadata|ovs|dhcp|l3]-agent
- /usr/lib/ocf/resource.d/fuel/mysql
- /usr/lib/ocf/resource.d/ocf/haproxy

The workflow of the scripts includes:

1. MySQL agent starts.
2. HAProxy and virtual IP addresses are set up.
3. Open vSwitch, metadata, L3, and DHCP agents start as Pacemaker clones on all the nodes.

The MySQL high-availability script primarily targets to perform automatic failover with the minimum possible downtime. The script must have a working Corosync in which it forms a quorum of replication epochs and then selects a master node which has the newest epoch.

By default, a five minute interval is configured for every cluster member to boot and participate in the election.

Every node is self-aware, which means that if nobody pushes a higher epoch retrieved from Corosync, it will become a master.

Seealso

- man corosync.conf or Corosync documentation
- Using Rules to Determine Resource Location

Plan the network

In this section you can find the description of architectural and configuration aspects to be taken into account when planning your network environment. Such as default logical networks and toplogies that Fuel deploys, as well as a possibilities to extend standard network functionality.

This section includes the following topics:

Logical networks

This section describes default (predefined) networks and their behaviour. Fuel enables you to create and modify networks using API, as well as modify service to networks mapping using networking templates.

Fuel deploys the following networks:

Public network

Virtual machines communicate with each other and access the Internet through Public network. Public network provides connectivity to the globally routed address space for VMs. The Public network IP address assigned to the network and compute (in case of DVR) nodes is used by Source NAT to enable the outgoing traffic from VM instances access the Internet.

Public network also provides Virtual IPs for public endpoints that are used to connect to OpenStack services APIs.

Finally, Public network provides a neighboring address range for floating IPs that are assigned to individual VM instances by the project administrator.

For security reasons, isolate Public network from other networks in an OpenStack environment.

Internal networks

Internal network is a general term for all networks in your OpenStack environment except for Public and Private network. Internal networks include Storage, Management, and Admin (PXE) Fuel networks. Internal network connects all OpenStack nodes within an OpenStack environment. All components of an OpenStack environment communicate with each other using internal networks. The internal network can also be used for serving iSCSI protocol exchanges between compute and storage nodes.

Do not confuse internal network with Private, as the latter is only related to a network within a project that provides communication between project's VMs. Isolate internal networks from Public and Private networks for security reasons.

Admin (PXE) network (Fuel network)

The Fuel Master node uses Admin network to provision and orchestrate the OpenStack environment. It provides DNS, DHCP, and a gateway to Slave nodes before the nodes are provisioned. Since Fuel Slave nodes obtain their network configuration from the Fuel Master node using DHCP, verify that Admin (PXE) network is isolated from all other networks in an OpenStack environment and use only the DHCP server that the Fuel Master node provides.

> Note
>
> Admin (PXE) network must always be untagged. Even when it is combined with other networks on the same network interface.

Storage network (Storage replication)

Storage network handles replication traffic from Ceph or Swift. Ceph public traffic is dispatched through br-mgmt bridge (Management network) by default. You can modify this setting through network templates.

Management network

Management network serves all other internal traffic such as database queries, AMQP messaging, and high-availability services, as well as iSCSI traffic between the compute and storage nodes.

Private network

For VLAN segmentation, the Private network is used for VLAN-separated project traffic. The Private network defines L2 topology by mapping a br-prv bridge to a node's interface, as well as determines the VLAN ID range.

For VXLAN/GRE segmentation, the Private network is used for tunneling project's traffic. The Private network defines L2 topology by mapping a br-mesh bridge to a node's interface and setting optional VLAN ID. It also specifies a tunnel ID range and network L3 parameters such as CIDR and gateway.

Networking Templates

You can use networking templates to create flexible network configurations. Networking templates enable you to:

- Create additional networks. For example, you can create a separate network for Swift storage traffic.
- Delete unnecessary networks.
- Add network roles.
- Create a network only if a relevant node role is present on the node.
- Customize networking topologies. For example, configure subinterface bonding.

However, networking templates have the following limitations:

- Interdependencies between templates for different node roles cannot be set.
- Mapping of network roles to networks and a network topology cannot be set for nodes individually. They can only be set for a node role or/and a node group.
- There is no web UI support for networking templates. You can only operate via CLI or API. After you upload a networking template, the Configure Interfaces tab in the Fuel Web UI becomes inactive.

> Note
>
> If you delete a template, Fuel automatically applies the default networking solution.

You can find the samples of network templates in the network template examples folder.

Specifying IP address ranges for logical networks

When you deploy an environment using Fuel, you can exclude specific IP addresses so that Fuel does not assign them. This helps to avoid network conflicts in case these IP addresses were previously reserved by other network entities.

To prevent IP address collisions, set the IP address range to be used by Fuel, for example, for nodes and VIPs, excluding the reserved IPs. In addition, you can specify multiple IP address ranges. If you have an IP address in use in the middle of the network IP address range, you can split the range to exclude the IP addresses in use.

Virtual IP addresses for load balancing

Fuel automatically allocates Virtual IP addresses (VIPs) for the load balancing service (LB). If automatic allocation does not meet your needs or you want to use an external load balancer, you can set an arbitrary IP address as a VIP through the Fuel CLI.

> Seealso
>
> - Fuel CLI Reference
> - Using Fuel CLI section of the Fuel User Guide

Multiple cluster networks

Mirantis OpenStack supports configuring multiple network domains per single OpenStack environment. With multiple network clusters you can deploy large scale clusters that match physical network layer of the data centre and provide a possibility to easily scale up your environment. Use this feature for environments that deploy a large number of Fuel slave nodes to avoid the broadcast storms that can occur when all nodes share a single L2 domain. You can configure multiple cluster networks with the help of Fuel for OpenStack environments that use neutron.

The multiple cluster network feature is based on node groups, which are groupings of nodes in an OpenStack environment:

- All logical networks are associated with a node group rather than the environment. The only exception is default Admin network. The default Admin network is shared between all environments and associated with all nodes that do not belong to any environment.

> Note
>
> When using the multiple cluster networks feature, you must set gateways for all networks within an OpenStack environment. Fuel generates static routes between networks of different node groups using these gateway addresses.

- Each node group belongs to an OpenStack environment.
- A default node group is created automatically when user creates a new environment. The environment can not have more than one default node group and the default node group cannot be deleted while its environment exists.
- Each default node group uses the same default Admin network. Other node groups, which are created by user through API, have their own Admin networks, which are not shared between environments.
- The default Admin network is shared between all environments. This network cannot be deleted. One Fuel installation always has exactly one default Admin network.
- The values for the default Fuel Admin (PXE) network are set using Fuel Menu. Default values for other networks are taken from release metadata.
- OpenStack environments deployed with Neutron support multiple node groups.

Nailgun manages multiple cluster networks as follows:

- A node serializes its network information based on its relationship to networks in its node group.
- When user adds a node to an OpenStack environment, Fuel automatically assigns the node to a node group based on the node's IP address. If the node's IP address does not fit into any Admin (PXE) network, then Fuel assigns the node to the default node group.
- Fuel automatically creates a default node group for a new environment. However, a user can also create a node group manually through API or CLI. In both cases, Fuel automatically generates a set of default networks for every new node group. Similarly, when you delete a node group, all networks which belong to this node group are deleted as well.

> Note
>
> The default Admin (PXE) network is not deleted, because it applies to a default node group of each OpenStack environment.

Additionally, a user can manually add or remove networks within a a node group through API or CLI.

- DHCP requests can be forwarded to the Fuel Master node using one of the following methods:
  - Configure switches to relay DHCP
  - Use a relay client such as dhcp-helper
- Fuel stores information about all node groups in the nodegroup table.

Seealso

- Viewing node groups in Fuel User Guide
- Fuel Release Notes

Network topologies

Fuel deploys network topologies using the OpenStack Networking service called Neutron.

Note

Since the upstream development efforts of Nova Network have been gradually reduced and Neutron is becoming a preferred option for a majority of OpenStack deployments, Nova Network is deprecated in Fuel. Use Neutron for all new deployments.

Neutron is a flexible network manager that enables you to create complex network configurations. Neutron provides both level 2 and 3 network virtualization, as well as IP address management (IPAM). In addition, neutron has multiple open-source and enterprise-class plugins that enable interoperability with such networking technologies as virtual switches and software-defined networking (SDN).

This section describes network topologies that have been thoroughly tested and are recommended for use in production environments. However, you can alternate these configurations to address the specific requirements of your cloud.

Fuel deploys the following network configurations:

- **Neutron with VLAN segmentation**

  In neutron's VLAN segmentation topology a VLAN is assigned to each tenant. IP subnets and ranges in different tenants can overlap. This is the default networking option in Fuel. The disadvantage of this option is that you must configure your networking equipment, as well as provide the total number of tenants, before configuring the network.

  Neutron with VLAN segmentation examples

	3 NICs	4 NICs
eth0	Port for Administrative network	Port for Administrative network

eth1	Port for the following networks: • Public/Floating (untagged) • Management (tag=102) • Storage (tag=103)	Port for the following networks: • Public/Floating (untagged) • Management (tag=102)
eth2	Port for Private network. The number of VLANs depends on the number of tenant networks with a continuous range.	Port for Private network with defined VLAN ID range
eth3	N/A	Port for Storage network

- **Neutron with tunneling segmentation**

  You can choose between VXLAN and GRE segmentation, with VXLAN being a default and preferred option for most of the OpenStack environments. GRE segmentation is deprecated in Fuel. In both VXLAN and GRE segmentations, project's traffic is isolated by encapsulating the traffic in tunnels. Both VXLAN and GRE segmentation are more flexible than VLAN in terms of the number of tenants. For example, VXLAN supports up to 16M channels. Network hardware configuration is significantly simpler compared to the VLAN segmentation and does not need to be synchronized with your L2 switch configuration. Both VXLAN and GRE support subnet overlapping in different tenants. However, the disadvantage of using GRE segmentation is that GRE encapsulation decreases the network speed between the instances, as well as increases the CPU usage on the compute and controller nodes.

  > Note
  >
  > You can configure GRE segmentation using CLI while VLAN and VXLAN options can be selected using UI wizard.

  Neutron with VXLAN/GRE segmentation examples

	2 NICs	3 NICs	4 NICs
eth0	Untagged port for Administrative network	Untagged port for Administrative network	Untagged port for Administrative network
eth1	Port for the following networks: • Public/Floating (untagged) • Management (tag=102) • Storage (tag=103)	Port for the following networks: • Public/Floating (untagged) • Management (tag=102)	Untagged port for Management network

eth2	N/A	Untagged port for Storage network	Untagged port for Public/Floating network
eth3	N/A	N/A	Untagged port for Storage network

Although Neutron with VLAN is the default option in the deployment wizard, some environments require other network topologies. Therefore, you must select the option that conforms with your configuration. For example, if you want to use a software-defined network (SDN) in your OpenStack environment, you must use Neutron with VXLAN tunneling segmentation.

Additionally, if you use VMware vCenter as a hypervisor, you must use the ML2 driver Fuel plugin.

Seealso

- Fuel CLI Reference
- vSphere integration in Fuel Installation Guide

Additional networking functionality

Fuel has plugins that provide additional networking functionality. The plugins enable such functionality as Software Defined Network, Firewall-as-a-Service, VPN-as-a-Service, support for different network adapters, and so on.

Additionaly, you can leverage Network Function Virtualization (NFV) in your OpenStack environment by enabling the following features:

- Guaranteed resources for workloads
- Huge Pages
- NUMA/CPU pinning
- SR/IOV
- Anti-affinity groups

Seealso

- Fuel Plugins
- Mirantis OpenStack NFVI Deployment Guide

Known limitations

- Fuel automatically configures the admin_floating_net and admin_internal_net networks for projects, as well as corresponding subnetworks. Use the neutron subnet-show command to view subnets. For example:

```
neutron subnet-show admin_floating_net__subnet
neutron subnet-show admin_internal_net__subnet
```

> Note
>
> For security reasons, Fuel does not configure a DHCP server for the admin_floating_net__subnet. If you use the admin_floating_net network for an instance boot request, select an additional subnetwork with a DHCP server enabled. For example, use admin_internal_net__subnet. Otherwise, the instance fails to boot. You must also manually configure the interfaces from the subnetwork without a DHCP server so the instance can obtain an IP address.

- Neutron will not allocate a floating IP range for your projects. After each project is created, a floating IP range must be created. This does not prevent Internet connectivity for a project's instances, but it would prevent them from receiving incoming connections. As an administrator, assign floating IP addresses for the project proceeding with the following steps:

  1. Get admin credentials:

     ```
     source /root/openrc
     ```

  2. Get admin tenant-ID:

     ```
     keystone tenant-list
     ```

     System response

     ```
     +----------------------------------+----------+---------+
     |                id                |   name   | enabled |
     +==================================+==========+=========+
     | b796f91df6b84860a7cd474148fb2229 |  admin   |   True  |
     +----------------------------------+----------+---------+
     | cba7b0ff68ee4985816ac3585c8e23a9 | services |   True  |
     +----------------------------------+----------+---------+
     ```

  3. Create one floating IP address for the admin project:

```
neutron floatingip-create --tenant-id=b796f91df6b84860a7cd474148fb2229 admin_floating_net
```

Plan the storage

There are different storage types that you need to consider for your workload requirements. This section provides the information on how to plan different types of storage for your OpenStack environment.

This section includes the following topics:

Storage components overview

The two fundamentally different storage types are:

- Ephemeral storage — This is a temporary storage for the operating system in a guest VM. Ephemeral storage is allocated for an instance in the OpenStack environment. As its name suggests, the storage will be deleted once the instance is terminated. This means that the VM user will lose the associated disks with the VM termination. Note that ephemeral storage persists through a reboot of the VM.
- Persistent storage — In contrast to the ephemeral storage, the persistent one exists outside an instance. Persistent storage is always available.

The Nova Compute service manages ephemeral storage.

- By default, ephemeral drives are stored locally on Compute nodes, in the Virtual Storage partition.
- If Ceph is configured for the environment and the Ceph RBD back end for ephemeral drives is enabled, Nova-compute stores ephemeral drives in Ceph.
- Other storage options are possible, such as an NFS share that is mounted from a SAN.

With Fuel deployment, you have the following storage options:

- Default providers — These are LVM for Cinder, local device for Swift, and Swift for Glance.
- Ceph — A storage platform that provides unified object, block, and file storage.

Plan block storage

The OpenStack component that provides the software to create block storage for your cloud is called Cinder. To configure block storage Cinder must be combined with one of the supported back ends.

The block storage back ends include:

- Cinder LVM (default) — each volume is stored as a logical volume in an LVM volume group on one of your Cinder nodes.
- Ceph — each volume is stored as an object in the Ceph RADOS object storage system.

> Note
>
> If you use vCenter as a hypervisor, you must use the VMDK driver to store your volumes in the vCenter datastore.

Choose between Cinder LVM and Ceph for the Cinder storage backend based on the following:

- Ceph provides a single shared pool of storage nodes for image storage.
- Ceph provides object replication capabilities by storing Cinder volumes as Ceph RBD objects. Each Ceph RBD object is stored as multiple RADOS objects. Ceph ensures that each replica of an object is stored on a different node. This means that your volumes are protected against hard drive and node failures.

  You can customize the Ceph data replication rules in the CRUSH map separately for each object pool, modify the number of object replicas, add different types of failure domains, and so on.
- LVM provides much less protection of your data than Ceph does. Even if you use RAID on each Cinder node, your data is only protected against a hard drive failure. If you lose the Cinder node, you will also lose all volumes on the node.
- Ceph consumes more disk space than LVM. LVM stores a single replica of the data, whereas Ceph stores at least two copies of your data so that your actual raw storage capacity must be two to three times bigger than your data set. You can however implement erasure coding striping to reduce the data multiplication requirements of Ceph.
- Ceph provides multi-node striping and redundancy for block storage.
- If you combine Ceph RBD backends for Cinder and Glance, you gain an important advantage over Cinder LVM: copy-on-write cloning of Glance images into bootable Ceph volumes.
- Ceph supports live migration of virtual machines with ephemeral drives, whereas LVM only supports live migration of volume backed virtual machines.

With Cinder LVM, you have the following configuration options:

- Let Fuel create a JBOD partition that spans all the storage drives in a node.
- Join all drives into a RAID array before deployment and have the array appear to Fuel as a single block device.

When deploying Ceph, Fuel partitions the Ceph storage nodes so that most of the space is reserved for Ceph-OSD storage. All other partitions for the node consume a portion of the first drive. To improve system performance, you can configure one or two SSDs and assign the "Ceph General" role to them in the Fuel web UI.

Seealso

- Ceph Hardware Recommendations

Plan object storage

Mirantis OpenStack supports Ceph as an object storage for applications.

Ceph includes the optional Ceph Object Gateway component called RADOS Gateway that applications can use to access RGW objects.

Note that the radosgw implementation of the Swift API does not implement all operations.

Ceph RBD uses RADOS directly and does not use the Swift API. This makes it possible to store Glance images in Ceph and still use Swift as the object store for applications.

Note that it is not possible to have both radosgw and Swift running in the same OpenStack environment, because the Ceph Object Gateway replaces Swift as the provider of the Swift APIs.

Storage plugins

Fuel integrates with leading storage providers and enables you to use third-party enterprise-class storage solutions as Cinder back end. If your organization uses an established storage platform, you can continue to leverage its benefits by using one of the Fuel plugins. Mirantis is constantly extending the list of the supported third-party storage platforms to address the requirements of its customers and partners.

For more information, see the Fuel plugin catalog.

Plan the monitoring tools

This section provides recommendations on planning the tools to monitor your OpenStack enviroment.

This section includes the following topics:

Monitor the OpenStack environment using the OpenStack Telemetry service

Fuel can deploy the Telemetry service called Ceilometer in your OpenStack environment. Ceilometer is an OpenStack component that collects and shares measurement data. You can use this data for:

- Billing purposes
- Creating alarm rules and using alarms for your purpose, including autoscaling with Heat

The billing process contains the following steps: metering and rating. Ceilometer covers only the metering part. For that purpose, the project collects and stores information about the system in form of samples to provide data for any metric that can be billed.

Ceilometer includes an alarming service that allows creating alarm rules that will be applied to the collected data. In addition, any system may use Ceilometer API to retrieve data.

Ceilometer has two sources of data:

- Polling
- Notifications from OpenStack services

Ceilometer collects the following types of data:

Metric samples

Metrics analyze activities in the cloud environment at the moment of sampling. Now, this information is gathered mostly by polling and may consume significant amounts of I/O processing resources and a large amount of database storage. Historically, metric samples were derived from OpenStack notifications as well, therefore, the term non-metric meter was introduced to describe these metrics. Also, the configuration option disable_non_metric_meters was added. For example, instance and disk are non-metric meters because they mean the existence of resources. When you choose the value for disable_non_metric_meters, you need to keep in mind that if it is True, you will not be able to retrieve non-metric meters using Ceilometer statistics API. It means that it would be impossible to know the amount of resources (images, instances) during a time interval. In Mirantis OpenStack, this configuration option is set to False.

To find out what metrics will be collected, refer to at least three files: meter.yaml, pipeline.yaml, and ceilometer.conf. In Mirantis OpenStack, custom changes were made to all these files to prevent performance degradation on one hand, and not to loose important metrics on the other.

Events

An event is a configurable Ceilometer structure. It is based on notifications triggered by services when various events take place in the OpenStack system. For example, "instance X was created" and "volume Z was deleted". Though the system sends these notifications

continuously, in the default configuration, monitoring of events uses less resources of the cloud environment than monitoring of meters. Polling has nothing to do with events in Liberty. The main file where events are configured is event_definition.yaml.

Metrics and events can be configured for handling certain meters and notifications depending on the information you are interested in.

You can configure Ceilometer to collect either a small or large amount of metering data. When collecting a large amount of data, Ceilometer processes high volume of database writes. For example, with 100 resources (virtual machine instances) and polling interval set to 1 minute, Ceilometer collects around 150000 samples per hour.

Seealso

- Settings tab in the Create a new OpenStack environment section of the Fuel User Guide
- Related projects in the Configure your environment section of the Fuel User Guide

Custom transformed metrics

Ceilometer provides the Transformers feature that you can use to create custom metrics based on the existing ones. By default, Ceilometer configures several transformer-based metrics, such as cpu_util, disk.bytes.rate, and others.

The most commonly used transformer-based metrics, such as cpu_util, disk.*.rate, and network.*.rate are implemented for the libvirt hypervisor by default. The Ceilometer compute agents create these metrics on the compute nodes. Ceilometer collects the metrics like other default metrics with the generic pipeline.yaml configuration and without performing the transformation on the notification agent side.

Important

Mirantis OpenStack supports all default metrics and does not support custom transformers which require cache. Only the unit_conversion and arithmetic transformers are supported as custom ones. Do not use the arithmetic transformer with an expression that contains more than one metric.

Using the Redis plugin, you still may configure transformers that require cache, but note that this approach is not performant and reliable enough and may lead to samples loss.

Seealso

- Transformers

Plan the OpenStack Telemetry service back end

Mirantis OpenStack defaults to installing MongoDB as the recommended back-end database for the OpenStack Telemetry service. The Fuel Master node enables you to choose the installation of MongoDB as a role onto a node. This resolves the Ceilometer performance issues caused by the volume of concurrent read/write operations. For instructions, see the Assign a role or roles to each node server section in the Fuel User Guide.

When planning your resources, consider the following:

- The MongoDB partition requires at least 10240 MB of free space to be allocated for internal use in replica mode.
- The resources consumed by metrics sampling are determined by:
  - The polling interval
  - The number of metrics being collected
  - The number of resources from which the metrics are collected (for example, how many instances are running, volumes, and so on)
  - The number of configured events

  The amount of storage required is also affected by the frequency with which you offload or purge the data from the database.
- The Working set for the MongoDB database should stay in memory to achieve good performance. The Working set is a part of data that is frequently used by Ceilometer.
- It is possible to have swap space on the nodes especially if MongoDB is deployed on controller nodes, where situations with extreme memory constraints or simultaneous memory usage by different services are possible.
- Frequent polling yields a better picture of what is happening in the cloud environment and also significantly increases the amount of data being processed and stored.

  Example:

  In one test sampling, the same metrics for the same fairly small number of resources in the same environment resulted in the following:

  - 1 minute polling accumulated 0.8 TB of data over a year.
  - 30 second polling accumulated 1.6 TB of data over a year.
  - 5 second polling accumulated 9.6 TB of data over a year.
- Ceilometer consumes fairly small amounts of CPU. However, the I/O processing is extremely intensive when the data is written to the disk. Therefore, we recommend using dedicated MongoDB nodes rather than running the MongoDB role on the Controller nodes.

  In our lab tests, nearly 100% of the disk I/O resources on the Controller nodes were sometimes consumed by Ceilometer writing data to MongoDB when the database was located on the Controller node and a small polling interval was used. This configuration halted or interfered with all other OpenStack services on the Controller node and prevented other processes from running.

- Ceilometer has the metering_time_to_live and events_time_to_live configuration options for samples and events correspondingly. By default, these options are set to 604800 seconds (seven days). It means that samples and events which are older than seven days will be removed with the native MongoDB job.

Seealso

- Optimize the MongoDB database
- Modify MongoDB indexes

Optimize the MongoDB database

To optimize the use of the MongoDB database, the OpenStack Telemetry service creates a set of single-filed indexes in the resource, meter, and event collections, as well as an additional compound index for statistic requests in the meter collection. The indexes improve performance by ensuring that the database does not scan the whole collection of data.

To optimize the MongoDB database follow these recommendations:

- Improve the efficiency of MongoDB resources usage by deleting unused indexes, especially from the meter collection.
- Improve the performance of Ceilometer statistic requests, which is used in the auto-scaling feature, by removing unused fields from the statistic index. By default, Ceilometer creates a compound index with fields that can be used as filters in a statistic request. The fields include:
  - Mandatory counter_name, set as an index prefix field, and timestamp, which is always used with boundary interval limit.
  - Optional resource_id, user_id, and project_id are included in default_statistic_idx.

If some parameters are not used in a statistic request, you can remove such fields from the statistic index.

Seealso

- Modify MongoDB indexes

Modify MongoDB indexes

You can modify MongoDB indexes to improve performance of the OpenStack Telemetry service, as well as the MongoDB database.

To update indexes in MongoDB:

1. From the mongo shell, log in to a MongoDB database:

```
mongo [-u username] [-p password]
```

2. Drop the old index:

```
db.meter.drop('default_statistic_idx')
```

3. Create a new index:

```
db.meter.createIndex({counter_name: 1, resource_id: 1, user_id: 1, \
project_id: 1, timestamp: -1}, {name: '<custom_statistic_idx_name>', \
background: true})
```

The resource_id, user_id, and project_id parameters are optional and should be added only if used in statistic requests.

When creating a new index in large collections, set the background parameter to true to avoid locking all the write and read operations.

Plan a Hadoop cluster

If your business requirements include analyzing huge amounts of unstructured data, you may want to deploy a Hadoop cluster. To deploy a Hadoop or Spark cluster in your OpenStack environment, use the Sahara OpenStack program. This section describes the settings and components that you need to plan for your Hadoop cluster.

This section includes the following topics:

Node requirements

All Sahara processes run on the controller nodes. The entire Hadoop cluster runs in virtual machines on the compute nodes.

For successful Hadoop installation, you must deploy an OpenStack environment with at least one controller node for the Sahara control processes and at least one compute node to run virtual machines for the Hadoop cluster.

A typical Hadoop installation includes:

- 1 virtual machine for management and monitoring processes — Apache Ambari and Cloudera Manager.
- 1 virtual machine that acts as the Hadoop master node to run ResourceManager and NameNode.
- Virtual machines serving as the Hadoop cluster nodes, each of which runs NodeManager and DataNode.

You must have exactly one instance of each management and master processes running in the environment. Configure other components as required by your environment.

For example, you can run the NodeManager and DataNode in the same virtual machine that runs ResourceManager and NameNode. You can also run DataNode and NodeManager in separate virtual machines.

Sahara communicates with object storage through Swift API. You can use Swift or Ceph with RadosGW as an object storage back end.

> Note
>
> If you have configured the Swift public URL with SSL, Sahara will only work with the prepared Sahara images, regardless of Swift usage. You can download the prepared images from Rackspace CDN.

Hardware requirements

Minimum hardware requirements for the OpenStack Hadoop cluster to run health check tests:

- **Controller nodes:**
    - CPU: 2
    - RAM: 6 GB

- **Compute nodes:**
    - RAM: 6 GB
    - CPU: 2

Limitations

An OpenStack Hadoop cluster has the following limitations:

- Do note use QEMU as a hypervisor to test Sahara.
- While deploying OpenStack environment using Fuel web UI, select a hypervisor other than QEMU.
- VirtualBox is not supported.

System prerequisites

Before deploying Sahara, verify that your environment meets system prerequisites.

Plugin Capabilities

The following table provides a plugin capability matrix:

Feature	Vanilla plugin	HDP plugin	Cloudera plugin	Spark plugin	MapR plugin
Neutron network	x	x	x	x	x
Cluster Scaling	x	x	x	x	N/A
Swift Integration	x	x	x	x	N/A
Cinder Support	x	x	x	x	x
Data Locality	x	x	x	x	N/A
High Availability	N/A	x	x	N/A	N/A

Floating IP addresses

Fuel configures Sahara to use floating IP addresses to manage virtual machines. Therefore, you must provide a Floating IP pool in each node group template you define.

Sahara assigns a floating IP address to each virtual machine, therefore, ensure that your Fuel configuration provides a pool of IP addresses to the cloud.

If you have a limited number of floating IP addresses or special security policies, you may not be able to provide access to all instances. In this case, you can use the instances that have access as proxy gateways. To enable this functionality, set the is_proxy_gateway parameter to true for the node group you want to use as proxy. Sahara will communicate with all other cluster instances through the instances of this node group.

> Note
>
> Set the Floating IP pool only to a proxy node group.

> Note
>
> Ambari and Cloudera Manager node groups must have a floating IP address.

Security groups

Sahara can create and configure security groups separately for each cluster depending on a provisioning plugin and the Hadoop version.

To enable security groups, set the auto_security_group parameter to True in all node group templates that you plan to use.

Virtual machine flavor requirements

A minimum of 4 GB of RAM is recommended for master virtual machines. Clouder and Hadopp master virtual machines require at least m1.large flavor.

For reasonable performance for workloads consider virtual machines with 8+ vCPU and 16+ GB of RAM.

Hardware-assisted virtualization

You must enable hardware-assisted virtualization for the hypervisor you use for OpenStack. Failure to enable this parameter may lead to frequent random errors during the deployment and operation.

While most modern x86 CPUs support hardware-assisted virtualization, its support still might be absent on compute nodes if they are themselves running as virtual machines. In that case hypervisor running compute nodes must support passing through hardware-assisted virtualization to the nested virtual machines and have it enabled.

Communication between virtual machines

Ensure that communication between virtual machines is not blocked.

Default templates

Sahara bundles default templates that define simple clusters for the supported plugins. Since the templates are already provided in the Sahara database, you do not need to create additional templates. Instead, use the default templates.

Supported default Sahara templates for plugins

You can use one of the following supported default Sahara templates with the corresponding plugins.

Template for Vanilla Hadoop 2.7.1

Category	Description
Template name	vanilla-2
Template description	The cluster template includes 1 master node and 3 worker nodes.
Number of node groups	2
Node group 1: vanilla-2-master	Includes the management Hadoop components: NameNode, HistoryServer, and ResourceManager. Also includes the Oozie workflow scheduler.
Node group 2: vanilla-2-worker	Includes the components required for data storage and processing: NodeManager and DataNode.

Template for Cloudera Hadoop Distribution (CDH) 5.4.0

Category	Description
Template name	cdh-5
Template description	The cluster template includes 1 master node, 1 manager node, and 3 worker nodes.
Number of node groups	3
Node group 1: cdh-5-master	Includes the management Hadoop components: NameNode, HistoryServer, and ResourceManager. Also includes the Oozie workflow scheduler.
Node group 2: cdh-5-manager	Includes the component that provides UI to manage Hadoop cluster: Cloudera Management.
Node group 3: cdh-5-worker	Includes the components required for data storage and processing: NodeManager and DataNode.

Template for Hortonworks Data Platform (HDP) 2.2, 2.3

Category	Description
Template name	hdp-2-x
Template description	The cluster template includes 1 master node and 4 worker nodes.
Number of node groups	2
Node group 1: hdp-2-x-master	Includes the management Hadoop components: Ambari, NameNode, MapReduce HistoryServer, ResourceManager, YARN Timeline Server, ZooKeeper. Also includes the Oozie workflow scheduler.
Node group 2: hdp-2-x-worker	Includes the components required for data storage and processing: NodeManager and DataNode.

Seealso

- Sahara documentation

Plan the orchestration

This section describes how to plan the resources integration and auto-scaling of your infrastructure using the Orchestration service (Heat).

Orchestration (Heat) overview

The Orchestration service, or Heat, implements a framework for managing the entire lifecycle of your infrastructure including auto-scaling inside the OpenStack cloud.

The Orchestration service uses human-readable Heat Orchestration Templates (HOT) in the declarative format to describe the deployment process of application infrastructure components. It enables you to automatically set up fully repeatable deployments. Orchestration service also supports the AWS CloudFormation template format through the OpenStack-native REST API and CloudFormation-compatible Query API.

When you deploy an OpenStack environment, the Orchestration service installs by default and integrates into Horizon.

The Orchestration service contains the following components:

heat-engine
: The main component of the Heat framework that is responsible for reading templates, launching instances, and providing events to the API users.

heat-api
: Provides a native REST API that processes API requests.

heat CLI
: Communicates with heat-api to process API requests.

heat-api-cfn
: Provides an AWS CloudFormation compatible API.

Seealso

- Heat project wiki page
- Heat developer documentation
- Mirantis blog post about Heat

Plan the vSphere integration

Fuel can deploy a Mirantis OpenStack environment with the VMware vSphere integration. This section provides an overview of the VMware vSphere support, as well as the details on how to plan the deployment of an environment with VMware vSphere as a hypervisor.

This section includes the following topics:

Overview

You can deploy your OpenStack environment using VMware vSphere as a virtualization platform. Deploying an OpenStack environment on top of VMware vSphere, you can get access to the unified OpenStack API and take advantage of such OpenStack services as the Data Processing service (BigData), Application Catalog service, and others.

Since VMware vSphere provides its own network capabilities, some OpenStack networking options are not supported. Your choice of network falls to either configuring a VMware Distributed vSwitch (DVS) or installing an NSXV Fuel plugin.

The VMware vSphere integrated environments relies on the VMware vSphere advanced features that include the following:

- vMotion workload migration
- vSphere High Availability
- vSphere Distributed Resource Scheduling (DRS)

You can easily access these features from the VMware vCenter driver provided by the VMware vSphere virtualization product family. The VMware vCenter driver makes management convenient from both the VMware vSphere Web client and the OpenStack Dashboard.

The VMware vSphere driver enables the interaction between the nova-compute service and VMware vCenter server. If the driver manages multiple ESXi clusters, Fuel enables specifying several clusters for a single OpenStack environment, so that a single nova-compute service manages multiple ESXi clusters through a single VMware vCenter server.

Unlike other hypervisor drivers that require the nova-compute service to be running on the same node as the hypervisor itself, the VMware vCenter driver enables the nova-compute service to manage ESXi hypervisors remotely. By default, Fuel places nova-compute on a controller node. However, if you plan to later extend this OpenStack environment with additional VMware vCenter clusters, place nova-compute that communicates with VMware vCenter on a dedicated compute-vmware node.

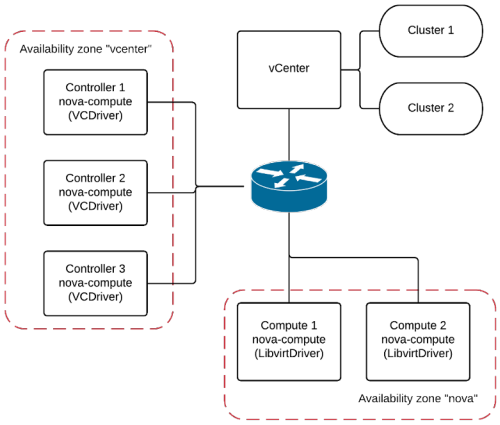
Seealso

- Install Fuel on VMware vSphere in the Fuel Installation Guide
- VMware DVS and VMware NSXv Fuel Plugins

Deploy an environment with VMware vCenter and KVM/QEMU

Fuel supports the dual hypervisor deployment feature that enables the cloud administrator to have a single point of control as well as a single pool of resources.

You can deploy an OpenStack environment with Vmware vCenter and KVM/QEMU using availability zones.



Seealso

- For the information on how to specify the availability zone in the Fuel web UI, see the VMware integration: Configuring a vCenter environment section of the User Guide.

Prerequisites

Before you deploy an environment integrated with VMware vSphere using Fuel, verify that the vSphere installation is up and running.

To configure VMware vSphere components, complete the steps described in the VMware vSphere prerequisites section in the Fuel Installation Guide.

Seealso

- VMware vSphere support in OpenStack
- Official vSphere documentation: Installation and Setup

Known limitations

This section describes known limitations and workarounds for environments that use VMware vSphere as a virtualization platform.

The limitations include:

- Fuel has been tested with VMware vCenter 5.5 and 6.0 only.
- The only supported image format is Virtual Machine Disk (VMDK).
- Generally, volumes that are created by the Block Storage service appear as SCSI disks. To be able to read and write to these disks, verify that the operating system of an instance supports SCSI disks. For example, a CirrOS image shipped with Fuel supports only IDE disks. Therefore, even if a volume is attached to an instance, CirrOS cannot use it.
- The Ceph back end is not supported for the Image service, Block Storage service, and RadosGW object storage.
- Red Hat Enterprise Linux compute nodes are not supported.

  If you plan to add RHEL compute nodes to your OpenStack environment, you cannot use VMware vCenter as a hypervisor.

Seealso

- The Preparing Murano images in VMware vCenter section in the Fuel User Guide

Plan the Fuel plugins

This section includes the following topics:

Fuel plugins overview

Fuel provides plugins that you can use to extend the functionality of your OpenStack environment and enable various third-party components and technologies.

Most of the Fuel plugins are developed by OpenStack community members, as well as by companies who support OpenStack. The Fuel plugins are distributed free of charge.

You can find the list of plugins validated by Mirantis to be compatible with the corresponding version of Mirantis OpenStack in the Fuel Plugins catalog.

> Note
>
> Mirantis recommends that you install all Fuel plugins before you deploy an OpenStack environment. Although, you can install some of the Fuel plugins after you deploy an OpenStack environment, it may require manual configuration. For more information, see plugin documentation.

Moreover, Fuel Plugins SDK enables you to develop any plugin that you need to meet your requirements.

> Seealso
>
> - Fuel Plugins SDK
> - Fuel Plugins catalog

Fuel plugin validation

Validation is the process that the Mirantis Partner Enablement team uses to ensure the provided plugin can be used with Mirantis OpenStack.

> Seealso
>
> - Fuel Plugin validation
> - Fuel Plugins SDK
> - Fuel Plugins catalog

Calculate hardware resources

After you read and understand what network, storage, and management topologies Fuel deploys, you can estimate resources that you will need to run your OpenStack workloads. This section provides examples on how to calculate the number of CPU cores and RAM per virtual machine, amount of storage, both persistent and ephemeral, network hardware, and IOPS.

When choosing the hardware on which you will deploy your OpenStack environment, plan the following:

CPU

Depends on the number of virtual machines that you plan to deploy in your cloud environment and the CPU per virtual machine. The amount of CPU greatly depends on the type of workloads that you plan to run in your environment. For example, environments used for heavy computational work may require more CPU than environments used primarily for storage.

Memory

Depends on the amount of RAM assigned per virtual machine and the controller node.

Storage

Depends on the local drive space per virtual machine, remote volumes that can be attached to a virtual machine, and object storage.

Networking

Depends on the network topology, the network bandwidth per virtual machine, and network storage.

This section includes the following topics:

Example conventions

For the purpose of example, we assume that your environment has the following prerequisites:

Environment prerequisites

Number of VMs	100
Amount of RAM per VM	4 GB RAM with dynamic allocation for up to 12 GB
Local storage per VM	150 GB
Persistent volume storage per VM	500 GB
Total persistent storage (object or block)	50 TB
Compute units	6 compute nodes with 2 sockets, 8 cores per socket, 2 GHz
Network requirements	• At least 100 Mbit/sec per VM • High availability • Latency-insensitive network storage

Calculate CPU

This section uses prerequisites listed in Example conventions.

When calculating CPU for compute nodes, you need to know the number of virtual machines you plan to run on each compute node.

This calculation presumes the following:

- No CPU oversubscription
- Use of hyper-threading
- CPU supports the technologies required for your deployment.

To calculate CPU:

1. Calculate the number of CPU cores per virtual machine using the following formula:

   ```
   max GHz/(number of GHz per core x 1.3)
   ```

   1.3 is the hyper-threading coefficient. If you do not use hyper-threading, use 2 instead.

   Example:

   ```
   16/(2.4 x 1.3) = 5.12
   ```

   Therefore, you must assign at least 5 CPU cores per virtual machine.
2. Calculate the total number of CPU cores:

   ```
   (number of VMs x number of GHz per VM)/number of GHz per core
   ```

   Example:

   ```
   (100 x 2)/2.4 = 84
   ```

   Therefore, the total number of CPU cores for 100 virtual machines is 84.
3. Calculate the required number of sockets:

   ```
   total number of CPU cores/number of cores per socket
   ```

   For example, if you use Intel® Xeon® Processor E5-2650-70 with 8 CPU cores.

   Example:

   ```
   84/8 = 10.5
   ```

   Round the result to the next whole number. Therefore, you need 11 sockets.

4. Round the number of sockets up to the next even number.

   For example, if the number of sockets is 11, then use 12.

5. Calculate the number of servers required for your deployment:

   ```
   total number of sockets/number of sockets per server
   ```

   Example:

   ```
   12/2 = 6
   ```

   Therefore, you need 6 dual socket servers.

6. Calculate the number of virtual machines per compute node:

   ```
   number of virtual machines/number of servers
   ```

   Example:

   ```
   100/6 = 16.6
   ```

   Round this result to the whole number. Therefore, you can deploy 17 virtual machines per server.

Using this calculation, you can add additional servers accounting for 17 virtual machines per server.

Calculate memory

This section uses prerequisites listed in Example conventions.

Using the example from the Calculate CPU section, calculate the amount of RAM a compute node will require to support 17 virtual machines.

When calculating RAM for a compute node, consider the memory a compute node itself requires to accommodate core operating system operations. Allocate at least 16 GB RAM or more for the core OS operations.

To calculate memory:

1. Use the following formula to calculate the total amount of RAM per compute node:

   ```
   amount of RAM per VM x number of VMs per compute node
   ```

   Example:

   ```
   12 x 17 = 204
   ```

Therefore, you need at least 204 GB RAM to accommodate the virtual machine workloads.

2. Add the RAM required to accommodate the core operating system operations:

   ```
   total amount of RAM per compute node + RAM for core OS operations
   ```

   Example:

   ```
   204 + 16 = 220
   ```

   Therefore, you need at least 220 GB RAM.

3. Round the amount of RAM you calculated in step 2 to the number that fits your server configuration and your selection of computer memory modules.

   For example, for a 2 CPU socket board that typically has 16 memory slots and 16 GB memory modules, you assign 256 GB RAM.

Adjust this calculation as required for your deployment.

Calculate storage

This section uses prerequisites listed in Example conventions.

When planning the number of disks and capacity required per server, you must consider storage for the following types of data:

- Ephemeral storage
- Persistent object storage
- Persistent block storage

When you select hardware for your compute nodes, understand the types of workloads you plan to process on them, as well as the impact of these workloads on storage. If you do not expect storage impact to be significant, then you may consider using unified storage. For example, a single 3 TB drive provides more than enough storage for seventeen virtual machines with 150 GB disk space. If speed is not important for your deployment, you might even consider installing two or three 3 TB drives and configure a RAID-1 or RAID-5 for redundancy. However, if speed is critical, you will likely want to have a single hardware drive for each VM. In this case, you may want to use a 3U form factor server with 24 drives. The backplane of the server must support the drive configuration that you plan to use.

You must also allocate disk space for the compute node operating system.

Calculate storage performance

This section uses prerequisites listed in Example conventions.

When estimating storage performance, the number of Input/Output operations per second (IOPS), throughput, and latency are the three important factors that you must consider.

Ephemeral storage

Use the following formula to calculate IOPS for the ephemeral storage based on the packing density:

```
drive IOPS X drives per server / VMs per server
```

However, the actual storage performance depends on the drive technology that you use. For example:

- If you use two mirrored 3.5" HDDs with 100 IOPS, then:

  ```
  100 X 2 / 17 = 12 Read IOPS, 6 Write IOPS
  ```

- If you use four 600 GB HDDs with 200 IOPS in RAID-10, then:

  ```
  200 X 4 / 17 VMs = 48 Read IOPS, 24 Write IOPS
  ```

- If you use eight 300 GB SSDs with 40K IOPS in RAID-10, then:

  ```
  40 000 X 8 / 17 = 19 000 Read IOPS, 9500 Write IOPS
  ```

Although, SSDs provide the best performance, the difference in cost between SSDs and the less costly platter-based solutions is significant. The acceptable cost burden is determined by the balance between your budget and your performance and redundancy needs. Also, the rules for redundancy in a cloud environment are different than in a traditional server installation, because entire servers provide redundancy as opposed to a single server instance being redundant. If you decide to use SSDs, you can increase performance even better by leveraging from the carefully planned and assigned read-optimized, write-optimized, or mixed-use SSDs.

In other words, the weight for redundant components shifts from individual OS installation to server redundancy. Therefore, it is far more critical to have redundant power supplies and hot-swappable CPUs and RAM than to have redundant compute node storage. For example, you have 18 drives installed on a server with 17 drives being directly allocated to each virtual machine. If one of the drives fails, then you can simply replace the drive and push a new node copy. The remaining virtual machines will process additional load that is present due to the temporary loss of one node.

Mirantis recommends that you use persistent block storage rather than ephemeral storage for virtual machines drives for most of the OpenStack environments. However, using local ephemeral storage may be beneficial for compute nodes that run performance-critical applications, such as databases.

Block and object storage

Depending on the type of servers that you use for your remote persistent block or object storage, the server configuration layout differs.

For example, for a 50 TB remote storage you may want to implement one of the following layouts:

- 12 drive storage server using 3 TB 3.5" mirrored HDDs drives
  - 36 TB raw, or 18 TB of usable space per 2U storage server
  - 3 storage servers (50 TB / 18 TB per server)
  - 12 slots x 100 IOPS per drive = 1200 Read IOPS, 600 Write IOPS per storage server
  - 3 servers x 1200 IOPS per storage server / 100 VMs = 36 Read IOPS, 18 Write IOPS per VM
- 24 drive storage server using 1TB 7200 RPM 2.5" drives
  - 24 TB raw, or 12 TB of usable space per 2U storage server
  - 5 servers (50 TB / 12 TB per server)
  - 24 slots x 100 IOPS per drive = 2400 Read IOPS, 1200 Write IOPS per storage server.
  - 5 servers x 2400 IOPS per storage server / 100 VMs = 120 Read IOPS, 60 Write IOPS per storage server.

The examples above are provided for your reference only. You can implement any layout that addresses the needs of your configuration. For example, a 36 drive server with 3 TB HDDs addresses the same requirements. However, Mirantis does not recommend that you use a single server in a production environment, because such configuration does not provide redundancy required by most of the enterprise-class deployments.

Calculate object storage

This section uses prerequisites listed in Example conventions.

Object storage typically protects data by keeping as many copies of the data as defined by the replication factor. By default, Fuel uses the replication factor of three for Swift and two for Ceph. For production environments, set the Ceph replication factor to three.

> Note
>
> You can use Ceph with the replication factor of two for testing deployments.

You must also accommodate the requirement for handoff locations. A handoff location is a partition or dedicated node on which Swift temporary places data in case of an error. This ensures three copies of the data are saved. Depending on the size of your deployment and the availability of storage resources, add 1 or 2 to the replication factor for handoff locations. Additionally, plan to scale out your object storage once it reaches 75% of capacity in order to avoid running out of space.

To calculate object storage:

1. Use the following formula:

```
(required capacity x ((replication factor + handoff location))/
percentage full
```

Example:

```
(50 x (3+2))/0.75 = 333 TB
```

Therefore, you need 333 TB or a multiplication factor of 6.66.

2. Optionally, adjust the multiplication factor.

   If the amount of storage is too high, you may want to lower the multiplication factor and plan to expand your storage in the future.

   Example:

   ```
   50 x 4 = 200 TB
   ```

   Therefore, with the multiplication factor 4, you need 200 TB.

3. Plan your storage server layout.

   Example:

   - 6 nodex x 12 3.5" 3 TB drives = 216 TB (36 TB per node)
   - 10 nodes x 10 2.5" 2 TB drives = 200 TB (20 TB per node)

     Mirantis recommends that you deploy more servers with fewer number of disks rather than fewer servers with more disks. In case of a storage failure, the size of data that Swift or Ceph will have to replicate is significantly smaller if you use fewer disks per server. Using fewer disks per server is also beneficial when you scale out your storage because of the amount of time required to re-balance your data objects. Therefore, in the storage layout examples above, first option is the preferred one.

Seealso

- Fuel Installation Guide

Calculate ephemeral storage

In many production environments, you may want to use a Ceph cluster or network storage for ephemeral volumes. You can also use any existing storage technology that is available in your environment.

To calculate ephemeral storage in a testing environment:

1. Use the following formula:

```
number of VMs per compute node x local storage per VM
```

Example:

```
17 x 150 GB = 2.55 TB
```

Therefore, you need a total of 2.55 TB of disk space on each compute node to accommodate the requirements of 17 virtual machines with 150 GB disk drives.

2. Add storage required to accommodate the compute node operating system storage requirements.

   For example, if speed is critical in your deployment, each compute node requires 18 disk drives, such as 200 GB SSD drives.

To calculate ephemeral storage in a production environment:

1. Use the following formula:

   ```
   total number of VMs x total ephemeral storage
   ```

   Example:

   ```
   100 x 150 = 15 TB
   ```

   Therefore, for 100 VMs with a requirement of 150 GB of ephemeral storage, you need 15 TB of storage in Ceph cluster or in network storage.

   To address ephemeral storage requirements, you can configure multiple SSD pools with disk drives with appropriate capacity. For example, you can use 400 or 800 GB mixed-use SSDs.

   If you use Ceph RBD as a back end for ephemeral storage, you may want to use SSDs for Ceph Object Storage Daemon (OSD) journal for better performance. Mirantis recommends that you use one small write-optimized SSD per five OSDs.

Seealso

- Calculate object storage

However, in many production environments you may want to use a Ceph cluster or network storage for ephemeral volumes. You can also use any existing storage technology that is available in your environment.

To calculate ephemeral storage in a testing environment:

1. Use the following formula:

```
number of VMs per compute node x local storage per VM
```

Example:

```
17 x 150 GB = 2.55 TB
```

Therefore, you need a total of 2.55 TB of disk space on each compute node to accommodate the requirements of 17 virtual machines with 150 GB disk drives.

2. Add storage required to accommodate the compute node operating system storage requirements.

   For example, if speed is critical in your deployment, each compute node requires 18 disk drives, such as 200 GB SSD drives.

To calculate ephemeral storage in a production environment:

1. Use the following formula:

   ```
   total number of VMs x total ephemeral storage
   ```

   Example:

   ```
   100 x 150 = 15 TB
   ```

   Therefore, for 100 VMs with a requirement of 150 GB of ephemeral storage, you need 15 TB of storage in Ceph cluster or in network storage.

   To address ephemeral storage requirements, you can configure multiple SSD pools with disk drives with appropriate capacity. For example, you can use 400 or 800 GB mixed-use SSDs.

   If you use Ceph RBD as a back end for ephemeral storage, you may want to use SSDs for Ceph Object Storage Daemon (OSD) journal for better performance. Mirantis recommends that you use one small write-optimized SSD per five OSDs.

Seealso

- Calculate object storage

Calculate network

This section uses prerequisites listed in Example conventions.

Information provided in this section is only relevant if you use Neutron without any software-defined network (SDN) solutions. Network calculation varies for each SDN.

This section includes the following topics:

Calculate network

By default, Fuel creates the following networks: Public, Storage, PXE (Admin), Management, and Private. However, your environment may require additional networks, such as multiple tenant private networks and so on. For many of the additional networks you must plan in advance and use VLANs.

To calculate network:

1. Calculate the speed per virtual machine that the network card that you selected provides:

   ```
   network card performance/number of virtual machines per server
   ```

   Example:

   ```
   10000/17 = 580 Mbit/s
   ```

   Therefore, one 10 Gb Ethernet network card provides 580 Mbit/s per virtual machine.

2. Adjust the number of network cards to the performance required for your deployment.

   For example, to accommodate the requirement of 100 Mbit/s per virtual machine, you need two 10 Gb Ethernet network cards. The two network cards also address the requirement of a highly-available network.

For the environment described in this section, you can use the following hardware configuration:

- 2 network switches per compute node, each with at least 12 ports for data: 2 x 10 Gb network interfaces per compute node x 6 compute nodes
- 1 x 1 Gb switch for IPMI (1 port per server x 6 servers)
- (optional) 2 Cluster Management switches

Calculate floating and public IP addresses

Before calculating floating and public IP addresses required for your OpenStack environment, read Network requirements.

For the purpose of examples, this section uses the following conventions:

Example conventions

Parameter	Value in this example
Number of controller nodes	3
Number of Zabbix nodes	1

Other nodes, such as compute, storage, or MongoDB	18 nodes in total: • 10 compute nodes • 5 Ceph OSD nodes • 3 MongoDB nodes
Number of virtual routers for all the tenants. The virtual routers must be connected to the external network.	10
Number of virtual machines that require direct access to the external network	100
Number of extra IP addresses. Typically, this value equals 3 and includes the following: • 1 virtual IP address for virtual router • 1 public virtual IP address • 1 IP address for default gateway	3

Calculate IP addresses for Neutron

This section uses conventions listed in Calculate floating and public IP addresses.

To calculate IP addresses for Neutron:

1. Calculate public IP range using the following formula:

   ```
   [(number of controller nodes + number of zabbix nodes) + number of extra IP addresses]
   ```

   Example:

   ```
   [(3 + 1) + 3] = 7
   ```

   Therefore, you need 7 IP addresses from the public IP range.

2. If you do not plan to use Distribute Virtual Routing (DVR), calculate floating IP range using the following formula:

   ```
   [number of virtual routers for all tenants + number of VMs]
   ```

   Example:

   ```
   [10 + 100] = 110
   ```

   Therefore, you need 110 IP addresses from the floating IP range.

3. If you use Distributed Virtual Routing (DVR) and plan to use floating IP addresses in the OpenStack environment deployment, use the following formulas:

- For VLAN segmentation:

  ```
  (number of controller nodes + number of extra IP addresses) + number of compute nodes
  ```

  Example:

  ```
  (3 + 3) + 10 = 16
  ```

  Therefore, you need 16 IP addresses.
- For GRE segmentation:

  ```
  (number of controller nodes + number of other nodes + number of extra IP addresses) + number o
  ```

  Example:

  ```
  (3 + 18 + 3) + 10 = 34
  ```

  Therefore, you need 34 IP addresses.

The following table summarizes results of the calculation:

Example conventions

Parameter	Value in this example	Use of public IPs	Use of floating IPs
Number of controller nodes	3	✓	--
Number of Zabbix nodes	1	✓	--
Other nodes	18	✓	--
Number of virtual routers for all tenants. The virtual routers must be connected to the external network.	10	--	✓
Number of virtual machines	100	--	✓
Number of extra IP addresses.	3	✓	--

Calculate IP addresses for Nova-network

This section uses conventions listed in Calculate floating and public IP addresses.

To calculate IP addresses for Nova-network:

1. Calculate the number of public IP addresses using the following formula:

   ```
   [(number of controller nodes + number of Zabbix nodes + number of other nodes) + number of extra IP
   ```

Example:

```
[(3 + 1 + 18) + 3] = 25
```

Therefore, you need 25 IP addresses from the public IP range.

2. Calculate the number of floating IP addresses.

   The number of floating IP addresses equals the number of virtual machines. Therefore, you need 100 IP addresses from the floating IP range.

The following table summarizes results of the calculation:

Example conventions

Parameter	Value in this example	Use of public IPs	Use of floating IPs
Number of controller nodes	3	✓	--
Number of Zabbix nodes	1	✓	--
Other nodes	18	✓	--
Number of virtual routers for all the tenants. The virtual routers must be connected to the external network.	10	n/a	n/a
Number of virtual machines	100	--	✓
Number of extra IP addresses.	3	✓	--

By default, Fuel creates the following networks: Public, Storage, PXE (Admin), Management, and Private. However, your environment may require additional networks, such as multiple tenant private networks and so on. For many of the additional networks you must plan in advance and use VLANs.

To calculate network:

1. Calculate the speed per virtual machine that the network card that you selected provides:

   ```
   network card performance/number of virtual machines per server
   ```

   Example:

   ```
   10000/17 = 580 Mbit/s
   ```

   Therefore, one 10 Gb Ethernet network card provides 580 Mbit/s per virtual machine.

2. Adjust the number of network cards to the performance required for your deployment.
   For example, to accommodate the requirement of 100 Mbit/s per virtual machine, you need two 10 Gb Ethernet network cards. The two network cards also address the requirement of a highly-available network.

For the environment described in this section, you can use the following hardware configuration:

- 2 network switches per compute node, each with at least 12 ports for data: 2 x 10 Gb network interfaces per compute node x 6 compute nodes
- 1 x 1 Gb switch for IPMI (1 port per server x 6 servers)
- (optional) 2 Cluster Management switches