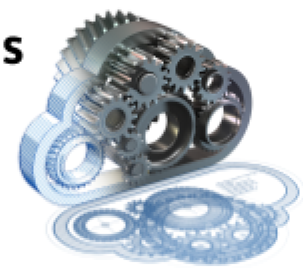
MIRANTIS
525 Almanor Ave, 4th floor
Sunnyvale, CA 94085
+1-650-963-9828 Phone
+1-650-968-2997 Fax

Reference Architecture for Compact Clouds

with Mirantis OpenStack 9.1 and Dell EMC Hardware



MIRANTIS DELL EMC

Table of Contents

Tables

Figures

1 Document Management

1.1 Trademarks

2 Overview

2.1 Executive Summary

This document provides a complete reference architecture and deployment guide for *Compact Clouds* with Mirantis OpenStack 9.1 (Mitaka) on Dell EMC compute and network hardware.

The *Compact Cloud* architecture is engineered to satisfy requirements for a small-scale, extensible OpenStack cloud platform for agile software development/QA/test, Continuous Integration/Continuous Delivery (CI/CD) and DevOps, with four-nines (99.99%) control plane uptime. To do this, *Compact Cloud* employs a virtualized, 'reduced footprint' control plane architecture — where OpenStack controller, OpenStack database, storage controller and monitoring components (on VMs) share a single hardware node ('infrastructure node'), and are made highly available with load balancing, clustering and messaging technologies.

The *Compact Cloud* specification includes OpenStack-native tools (e.g., Murano, Heat, etc.) for deployment automation, and specifies a complete toolchain (StackLight) for operations and performance data logging, monitoring, alerting, visualization and analysis.

2.2 Business Drivers for Compact Cloud

The *Compact Cloud* architecture for Mirantis OpenStack 9.1 on Dell EMC Hardware offers significant benefits to organizations developing software:

- ***Reduction of IT capital costs***
    - Use of open source software (OpenStack and all other components) eliminates licensing fees, providing savings that grow with scale.
    - Application of virtualized, distributed storage technology (Ceph) permits use of standard servers instead of purpose-dedicated storage hardware.
    - Use of software-defined networking (OpenVSwitch (OVS) SDN) constrains need for hardware switches, routers, and physical gateways.
    - Simple rack architecture, comprising Dell EMC top-of-rack (TOR) switches, server hardware powered by Intel, and select Intel mass storage devices, is easy to order, configure, and expand.
- ***Increased IT agility - reduced IT operating costs***
    - Automated full-stack provisioning for rapid deployment of new clusters and nodes.
    - Integrated monitoring, alerting and analytics simplify resource usage accounting, help ensure cloud performance and SLA compliance, and accelerate MTTR for issues.
    - Application catalog (Murano) enables rapid, self-service retrieval and deployment

of conventional and containerized apps and development environments.
    - OpenStack APIs, Heat orchestration and other tools accelerate cloud operations and make them repeatable.
- ***Faster software cycle times***
    - Software-defined cloud data center enables easy composition of reliable, repeatable, readily-accessible sandbox environments for development, QA and test.
    - DevOps (CI/CD) software pipeline, particularly when it exploits containers for service isolation and dependency management, accelerates release cadence by up to 4x (e.g., 6 releases per year vs. 24 releases per year). Deliver new features faster.

2.3 High-Level Requirements

The *Compact Cloud* architecture is engineered to:

- Accelerate pipelined development (from code check-in to go-live production) by 4X for applications and infrastructure (i.e., infrastructure as code, configuration as code).
- Provide 99.99% ('Four Nines') uptime — including high availability and disaster recovery (HA/DR) — for cloud-native apps, as well as monitoring, billing, backup/recovery, non-disruptive patching, and upgrades.
- Enforce IT security and regulatory compliance — including configuration management, identity management, and the ability to apply constraints automatically in the software build/test/deploy pipeline.

2.4 Sample Use-Cases for Compact Cloud

A *Compact Cloud* might be used to provide/enable:

Developer/QA Sandboxes - Custom, standardized virtual (or containerized) environment configurations, stored as code (e.g., deployment scripts, OpenStack HEAT templates, Vagrantfiles, Dockerfiles, Salt formulas, etc.) with associated, curated binary components, in multifunctional local or secure public repos. Sandboxes can be instanced/built/launched on demand, accessed conveniently with tools like SSH, RDP or VNC, and used to host workloads for in-line/smoke testing and debugging, functional testing (e.g., feature- and feature-group testing), and possibly non-functional testing (e.g., stress testing, load testing, volume testing). Instanced sandbox environments can be relinquished after use, freeing resources. They can also be shelved (halted for later rapid restart) and/or snapshotted (imaged and stored).

Commit Verification - When developers submit patchsets to version control, code must undergo syntax checks and unit tests against dependent modules. A range of standard tools is available to automate this process: running lint and similar tools, creating required virtual environment(s), building and deploying the app, running unit tests, and passing or rejecting, then relinquishing resources (or, in some situations, keeping them accessible for examination). Due to the frequency of code-commits, commit-check mechanics must be reusable and as lightweight as possible.

Nightly Builds - Typically performed when shared resources are least occupied. The CI system (e.g., Jenkins) automatically performs a set of integration tests over the current, unreleased branch: creating a virtual environment, provisioning it with workload and sample data, running tests, and reporting results. Because build environments tend to have large resource footprints, they should be reusable across several test runs.

Release Verification/Staging - Before production rollout of a new version of a workload, the release needs to be verified in a virtual staging environment. Steps involve packaging up the source tree, assembling the staging environment and running required upgrade procedures, deploying workload components and data samples, and executing the test suite.

3 Compact Cloud Components and Node Roles

The OpenStack cloud framework comprises many components, each providing essential services like Identity (Keystone), virtualization management (Nova-compute), volume/block storage virtualization (Cinder), object storage (Glance), network virtualization management (Neutron), etc., and integrated via drivers/plugins with physical and virtual infrastructure (e.g., KVM hypervisor, OpenVSwitch (OVS) SDN, Ceph distributed storage). A working OpenStack cloud also uses a database (e.g., MongoDB) to maintain its state, plus additional components (e.g., RabbitMQ, HAproxy) for messaging, load balancing, failover, etc.

Mirantis OpenStack 9.1 (MOS) — a distribution of OpenStack release Mitaka — is a hardened, bug-fixed set of OpenStack components, drivers, select virtual infrastructure, database, load balancing/HA and other open source components — prescriptively selected, integrated and tuned, and supplied with tooling (Fuel) for rapid, simplified configuration and deployment.

MOS groups OpenStack and other required components into 'roles' — e.g., controller, compute, network, Ceph OSD, etc., which can reside on physical or virtual hosts (nodes). A single physical server can host several (non-conflicting) roles, hosted on VMs. The *Compact Cloud* architecture implements a 'reduced footprint' OpenStack 'infrastructure node' that hosts OpenStack controller components, storage controller components, OpenStack database (MongoDB), and monitoring components on four separate VMs, plus an optional VM containing the Fuel Master Node (deployment tooling). To achieve full control-plane high availability, this multi-function infrastructure node is duplicated across failure domains and its components made resilient using HAproxy (OpenStack component HA), Galera cluster (OpenStack database HA), and other technologies, as appropriate.

Major roles required for *Compact Cloud* are described in detail, below. Specific versions of components deployed depend on maintenance updates installed. This information may be found in the official MOS documentation.

3.1 Control Plane

The OpenStack control plane includes roles for cloud operations, database, messaging and high availability (HA). *Compact Cloud* achieves high performance and reliability and relatively small scale and expense by exploiting a 'reduced footprint' virtualized control plane architecture: placing OpenStack controller, OpenStack database, storage controller, and metrics database on individual KVM VMs that share a physical Infrastructure node. This Infrastructure node architecture is tripled to achieve high availability (HA), using HAproxy, Galera cluster and similar technologies. In terms of automation, Fuel's "Reduced Footprint" feature is used to deploy Infrastructure nodes and span required VMs on them.

3.1.1 Controller Role

The controller role includes components enabling management of coordinated cloud operations. Controller components present REST APIs that can be used directly, via SDKs in a wide range of language environments, via the Horizon web UI, or by other compatible cloud management tools. MOS controller components include:

- Nova-scheduler
- Nova-api
- Glance-registry
- Glance-api
- Keystone
- Cinder-api
- Ceilometer
- Sahara
- Murano
- Heat
- Horizon
- HAProxy
- Neutron-api
- OpenLDAP proxy (optional)

3.1.2 Database Role

Several OpenStack components require a database for storing configurations, states, etc. Two kinds of database roles are most important in MOS:

- ***OpenStack database role*** - database used by OpenStack components to store information (e.g. Keystone, Nova, Glance, Neutron, Cinder). In MOS, this role is normally performed by MySQL/Galera.

- ***Telemetry database role*** - database used by the Ceilometer component to store metrics collected from the cloud. In MOS, this role is normally assigned to MongoDB.

3.1.3 Messaging System Role

Most OpenStack services use AMQP implementations for message transport and RPC. In MOS, this role is normally performed by RabbitMQ.

3.1.4 Storage Controller Role

In MOS, Mirantis recommends using Ceph for all storage types (object, block, file). The storage controller role thus comprises:

- ***Ceph Monitor (ceph-mon)*** - which maintains maps of the cluster state, including the monitor map, the OSD map, the Placement Group (PG) map, and the CRUSH map.
- ***Ceph Rados Gateway (RadosGW)*** - a FastCGI module for interacting with a Ceph storage cluster, providing a Swift/S3-compatible API for object storage.

Please check official Ceph documentation to learn more about Ceph components.

3.1.5 Highly Available Control Plane

To maintain high availability (HA), the MOS control plane needs to be set up to avoid single points of failure (SPoF). Control-plane node roles (e.g., controller, OpenStack database, storage controller, monitoring database) reside on KVM virtual machines, duplicated across (a minimum of) three physical servers. The physical control plane servers are distributed across independent racks, and linked by redundant network connections. This layout ensures that an availability zone failure does not result in multiple controllers becoming inactive.

The MySQL/Galera cluster, which stores the current state of the OpenStack environment, is laid out in active-active mode across control-plane servers, to provide continued operation in case of instance failure.

Deployment of control plane nodes also needs to take into account the location and type of load balancers used, ensuring that a load balancer failure does not cause a service outage.

Each of the services housed on the controller nodes has its own mechanism for achieving HA:

- nova-api, glance-api, keystone-api, neutron-api and nova-scheduler are stateless services that 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, or a shared session cache.
- Galera provides active-active high availability for the database.

- Pacemaker cluster - HA and load balancing stack. Pacemaker relies on the Corosync messaging layer for reliable cluster communication. Corosync implements the Totem single-ring ordering and membership protocol. It also provides UDP- and InfiniBand-based messaging, quorum, and cluster membership to Pacemaker.

3.2 Data Plane

The OpenStack data plane does the heavy lifting of hosting workloads by virtualizing and orchestrating compute, network and storage resources.

3.2.1 Compute Role

The compute role (virtualization layer) comprises the following MOS components:

- KVM
- Nova-Compute
- Ceph Client
- Neutron OVS Agent
- Ceilometer Agent

3.2.2 Network Role

The Network Role comprises the following MOS components:

- Neutron L3 Agent
- Neutron DHCP Agent
- Neutron Metadata Agent
- Neutron OVS Agent

3.2.3 Ceph OSD Role

A Ceph OSD Daemon (Ceph OSD) stores data, handles data replication, recovery, backfilling, rebalancing, and provides some monitoring information to Ceph Monitors by checking other Ceph OSD Daemons for a heartbeat.

3.3 Cloud Networks

To provide robust, scalable, high-performance network connectivity and throughput, the *Compact Cloud* architecture uses the following network segments:

- ***PXE/Admin -*** This segment is used for discovering, provisioning, deploying, configuring and administering cluster members (nodes). No routing is necessary, although the Mirantis OpenStack node needs to be reachable for cloud infrastructure administration.
- ***Management -*** Cloud components communicate over the management network.
- ***SAN -*** This network is used for traffic among Ceph Clients and Ceph Monitors.
- ***Storage Replication -*** This network is used for internal Ceph replication traffic only. There is no need to make this network accessible from outside the cluster.
- ***Private -*** Contains the tenant-specific virtual networks. An SDN solution might take ownership of this aspect of OpenStack networking.

- ***Public/External -*** The public network contains the VIPs for Horizon, all service endpoints for cloud operator command-line tools, and the floating IP range to make instances accessible outside a tenant network.

3.4 Monitoring Component

Mirantis' StackLight Toolchain is an operational health and response monitoring solution for clouds (*Compact Cloud* being one example) built in compliance with Mirantis OpenStack reference architectures. StackLight collects information from the entire MOS deployment (nodes, services, components, interfaces, etc.) and processes this data so that it can be easily consumed and analyzed by end users. StackLight lets operators visualize three key aspects of MOS:

- **Metrics** - Measurements are taken from nodes, services and interfaces throughout the environment and metrics calculated and aggregated from these. These metrics can be visualised graphically, giving in-depth insight into the cloud's operational health and performance.
- **Events** - Events are collected from log files, OpenStack service notifications and other sources, and can be filtered, queried, and visualised to gain deeper understanding of specific issues or behaviors within the system.
- **Alarms** - Alarms are generated using rules configured in StackLight — applying these to collected logs and metrics.

3.4.1 Monitoring Role

The monitoring node aggregates the following components:

- ElasticSearch
- Kibana
- InfluxDB
- Grafana
- Nagios

4 Compact Cloud Reference Architecture

This chapter summarizes engineering goals, component selections and configuration requirements for Mirantis OpenStack *Compact Cloud* on Dell EMC hardware.

Table 1. Compact Cloud Requirements and Specifications, Summary

Component	**Cloud Configuration**
Controller High Availability	Standard MOS RA design and configuration for 99.99% (four-nines) control plane uptime
Hypervisor	KVM
Glance Backend	Ceph/RBD
Object Storage	Ceph/RadosGW
Cinder Backend	Ceph/RBD
Nova Storage	Ceph/RBD
Keystone API version	3
Keystone Identity Backends	● Mysql for domain 'default' ○ Openstack services user IDs ● LDAP, each Organizational Unit (OU) maps to a Keystone domain ○ Cloud end-user IDs
Keystone Assignments Backend	MySQL
Complementary Projects	● Heat ● Ceilometer ● Sahara ● Murano ● StackLight
NIC Bonding type	Linux bonding, LACP, VLT
Control Plane	Controller, OpenStack Database, Messaging system, and Network roles are joined into one 'Compact Controller' role

Table 1. Compact Cloud Requirements and Specifications, Summary - Continued	
Number of Small Controller Nodes	3, virtualized
Number of Compute Nodes	Up to 50 (minimum 1)
Number of Storage Controller Nodes	3, virtualized
Number of MongoDB Nodes	3, virtualized (minimum 1)
Number of Monitoring Nodes	3, virtualized (minimum 1)
Number of Ceph OSD Nodes	Up to 27 (minimum 4)
Number of Infrastructure Nodes	3
Neutron ML2 Backend	OVS + VxLAN
Neutron External Network	1, flat-mode
Neutron Complementary Services and Their Backends	None

4.1 Sizing

Controller node configuration and the number of controller nodes recommended for *Compact Cloud* (3+, can be virtualized for smaller implementations) is calculated to ensure stable and highly available service at a minimum of four-nines (99.99%) control-plane uptime.

Data plane (e.g., storage controller, monitoring, Ceph OSD) node configurations and recommended numbers (typically 3+ for key node types) are calculated in terms of performance, industrial best practices, and MIrantis' experience. The recommended deployment ensures that *Compact Cloud* will continue to run at full planned capacity (i.e., will comply with SLA) if one key data plane node fails.

4.2 Core Network Configuration

The Compact Cloud architecture employs a top-of-rack to leaf (ToR to Leaf) aggregation schema for core networking. This network configuration scales better than ring configurations, and is easier to build and maintain.

The number of ToR switches is chosen in order to enable all required connections to terminate within each rack. If racks have more than one ToR switch, these switches should be joined as a stack.

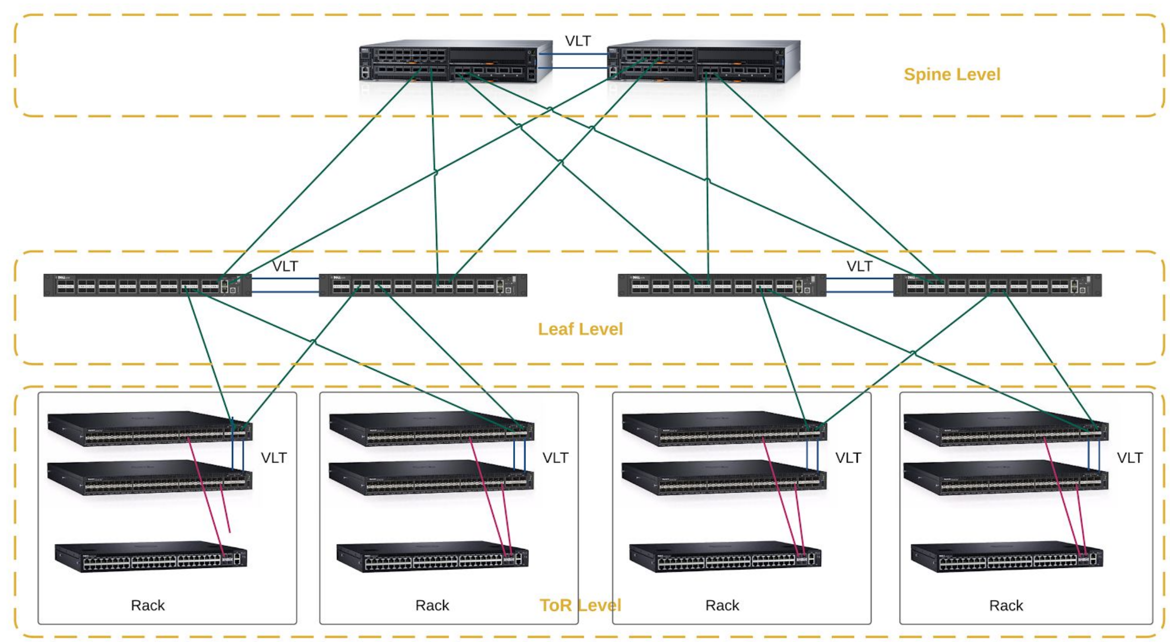


Figure 1. Compact Cloud core network diagram

On each level, each pair of switches forms a single Virtual Link Trunking (VLT) domain. At lower levels, each switch connects to two upper level switches within one VLT domain, providing full redundancy in case of link or switch failure.

4.3 High Level Network Diagram

All nodes are connected to two 10GE ToR switches (Dell Networking S4048-ON) by two interfaces joined into a Link Aggregation Control Protocol (LACP) group. This virtual interface is used for MOS networking. Each node is also connected to a 1GE ToR switch — one for the Admin/PXE FUEL network and for iDRAC.

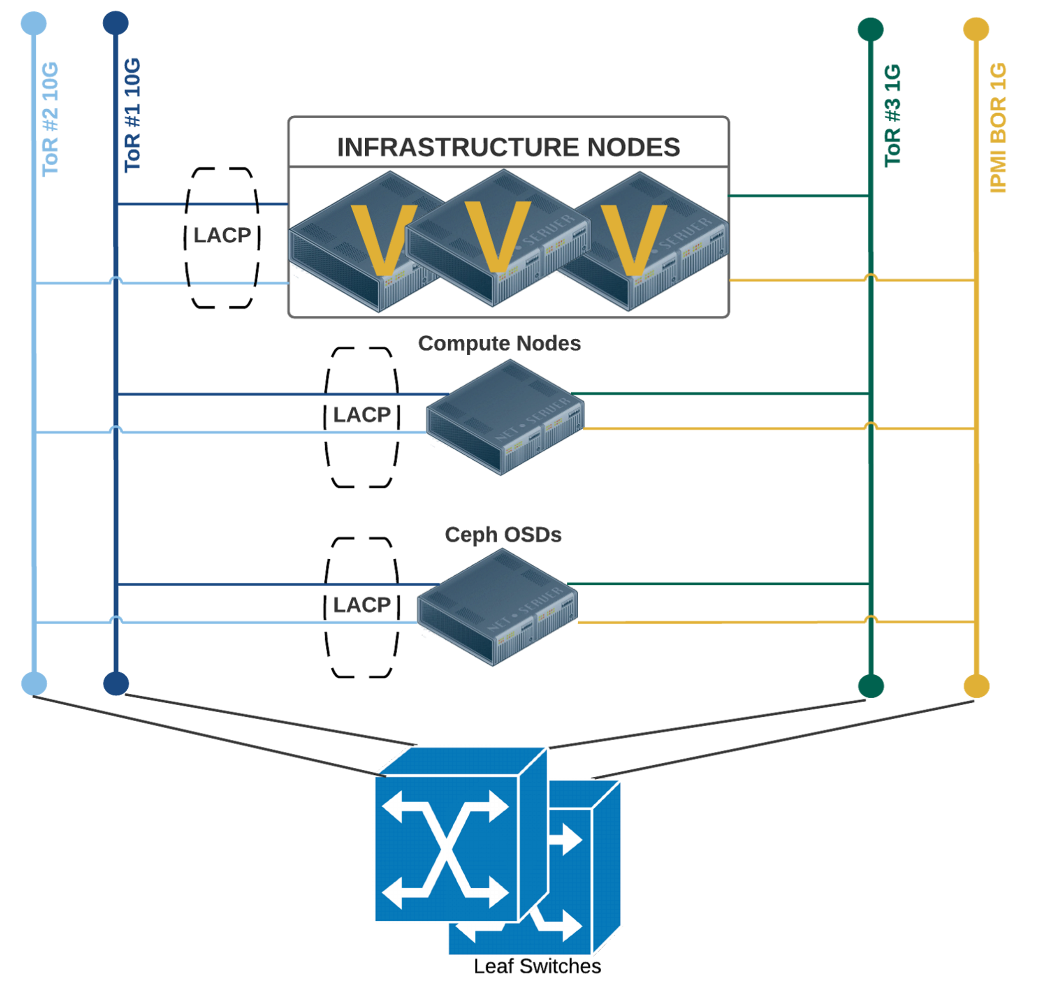


Figure 2. Compact Cloud high level network diagram

4.4 Node to Network Interconnections

4.4.1 Controller VM Network Connections

Controller VMs for *Compact Cloud* are equipped with five virtual NICs. Each vNIC connects one cloud network in untagged mode. Controllers are connected to Admin/PXE, Management, SAN, Private, and Public networks.

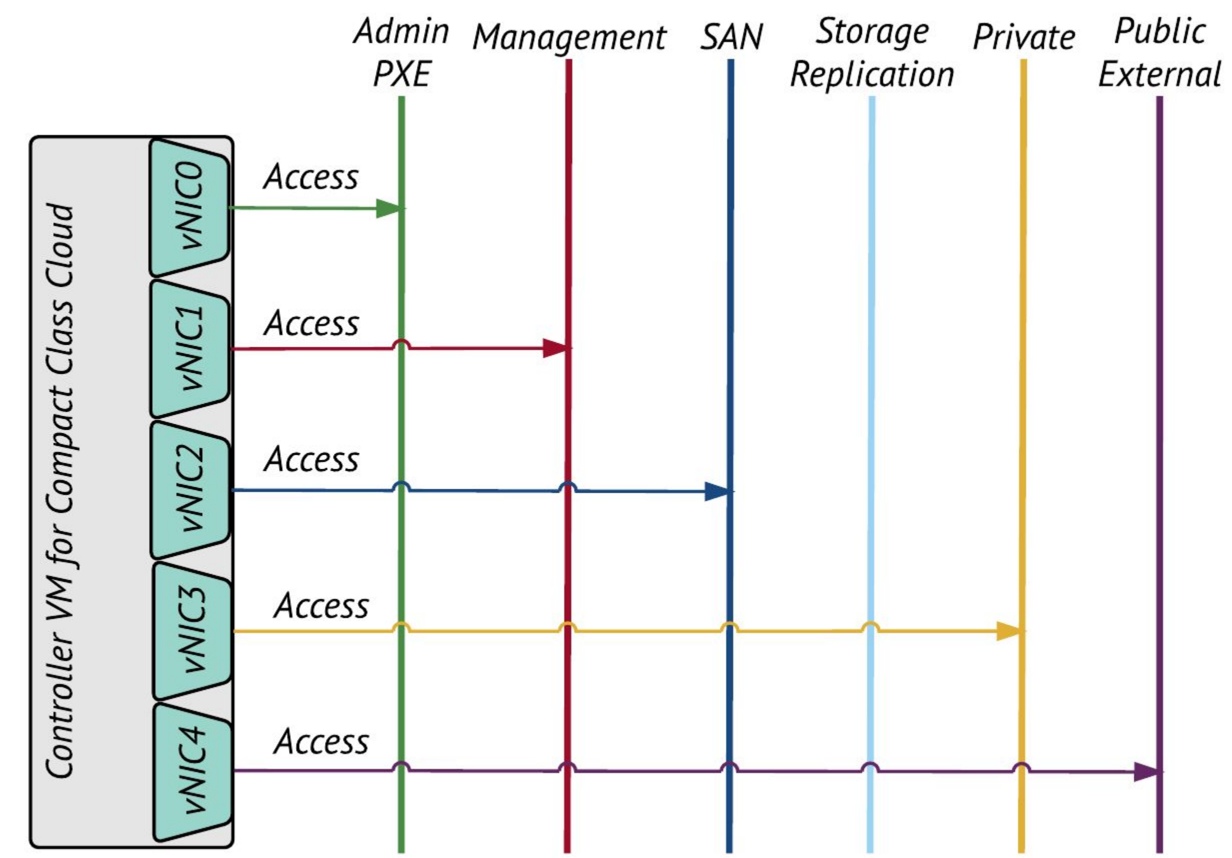


Figure 3. Compact Cloud controller to network interconnections

4.4.2 Compute Node Network Connections

Compute nodes for *Compact Cloud* are each equipped with three NICs. The first NIC is connected to the Admin/PXE network in untagged mode. The second and third NICs are bonded (using Linux bonding) into a single logical interface (LACP mode) that serves Managment, SAN, and Private networks on tagged mode VLANs.

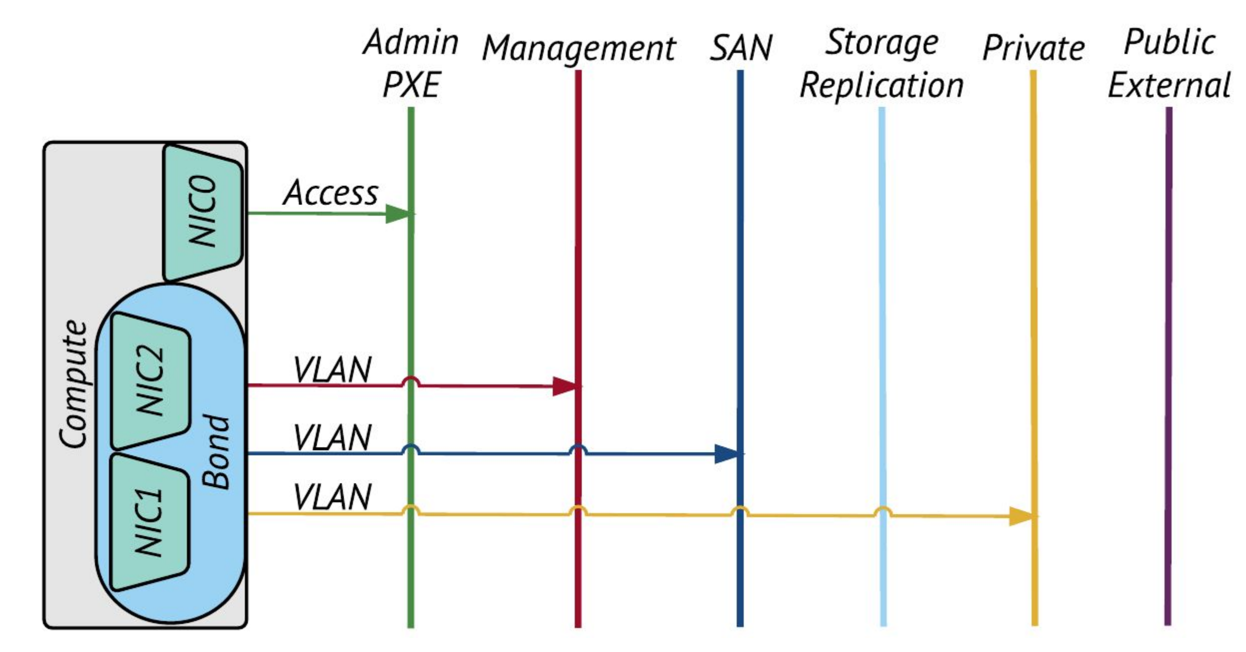


Figure 4. Compact Cloud - compute node to network interconnections

4.4.3 Storage Controller VM Network Connections

Storage controller VMs for *Compact Cloud* are equipped with four virtual NICs. Each vNIC connects with one cloud network — Admin/PXE, Management, SAN, and Storage Replication networks — in untagged mode.

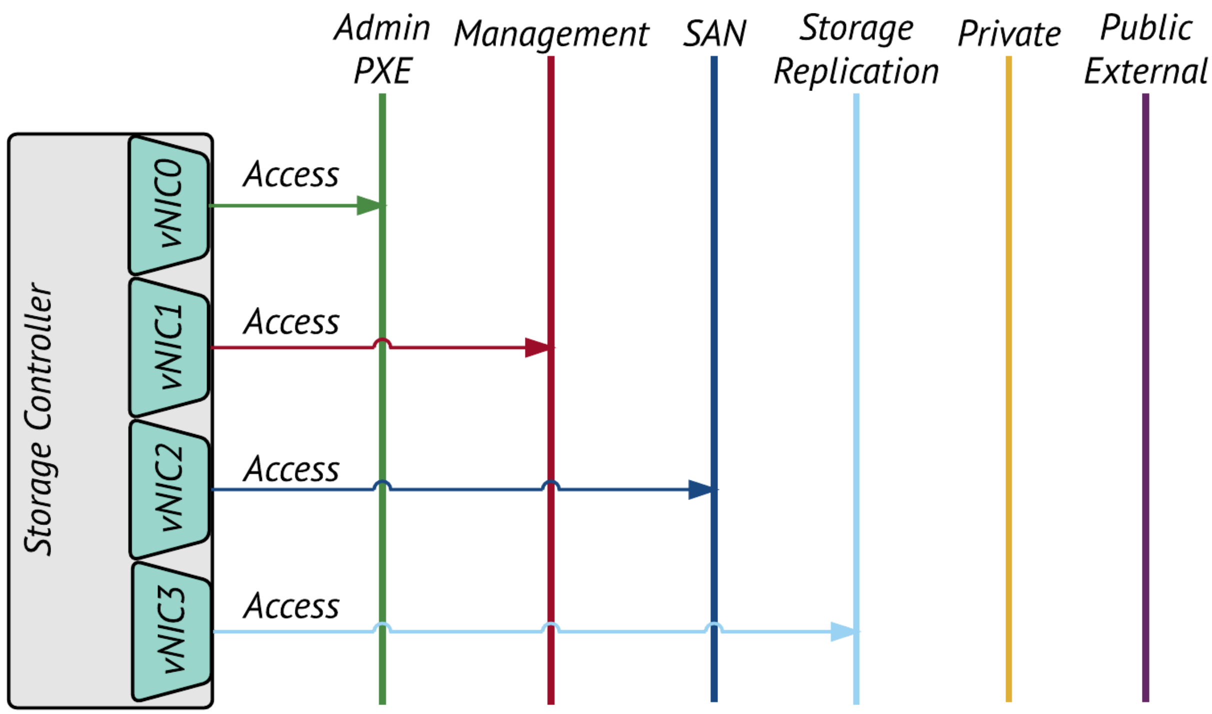


Figure 5. Compact Cloud - storage controller to network interconnections

4.4.4 Ceph OSD Network Connections

Ceph OSD nodes in *Compact Cloud* are equipped with five NICs. Though a three(3)-NIC configuration would be possible, Mirantis recommends separating the Storage Replication network on its own pair of NICs, making effective storage bandwidth predictable and eliminating a potential source of contention.

For cost efficiency, we recommend providing these five physical NICs by using one single-port NIC and two dual-port NICs. The single-port NIC is connected to the Admin/PXE network in untagged mode. The first ports of each dual-port card (effectively the second and fourth NICs) are bonded together (Linux bonding, LACP mode) to serve Management and SAN networks as tagged-mode VLANs. The second ports of each dual-port card (effectively the third and fifth NICs) are likewise bonded together to serve the Storage Replication network via a tagged-mode VLAN. This connection scheme addresses not only individual cable or switch failure but also chipset/NIC failure.

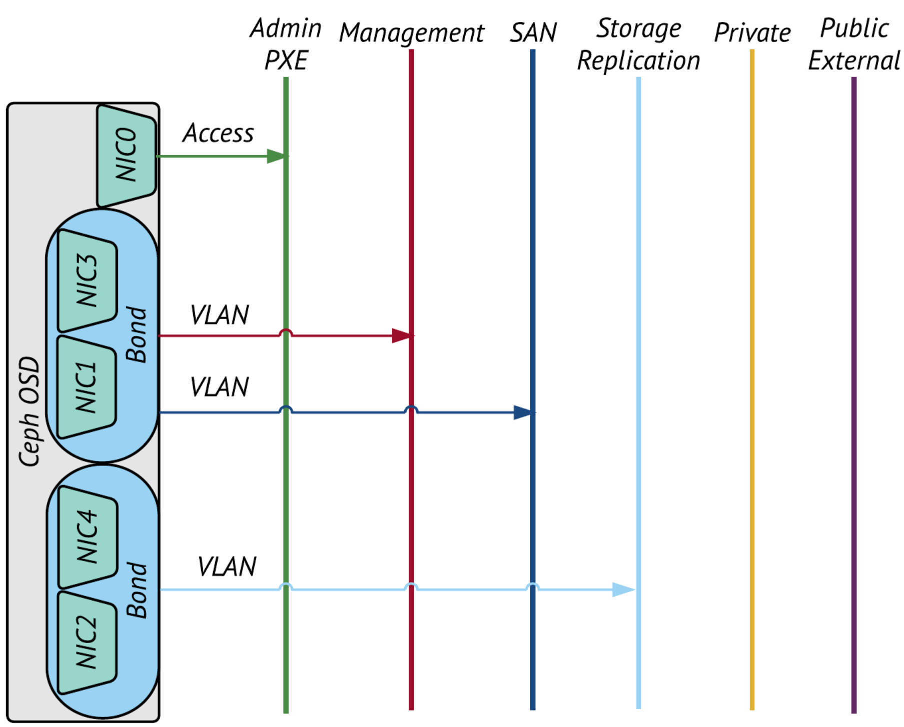


Figure 6. Compact Cloud - Ceph OSD node to network interconnection

4.4.5 Monitoring and MongoDB Node VM Network Connections

Monitoring and MongoDB nodes for *Compact Cloud* are equipped with two virtual NICs. The first is connected to the Admin/PXE network, the second to the Management network, both in untagged mode.

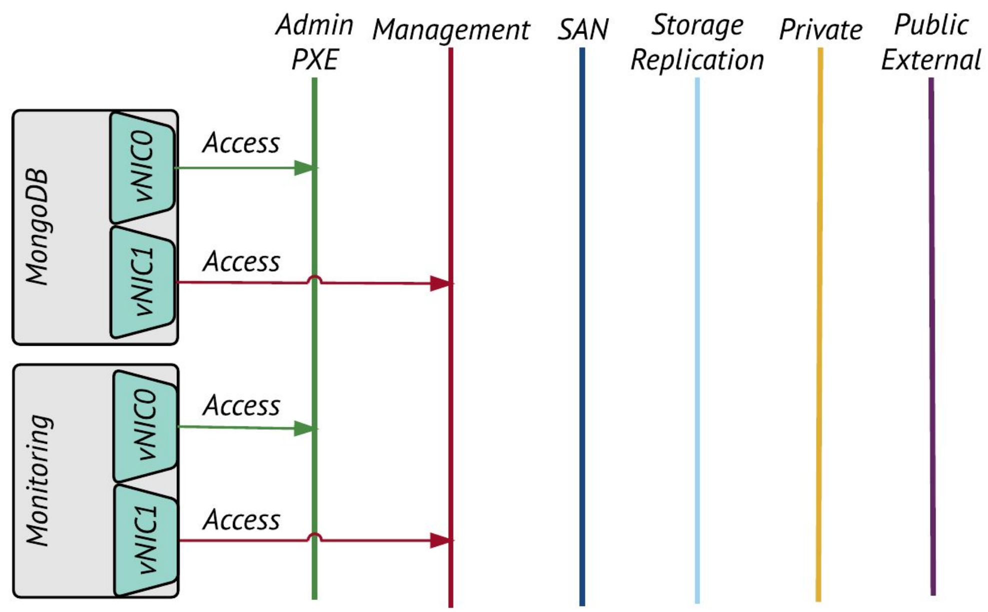


Figure 7. Compact Cloud - MongoDB and monitoring node to network interconnection

4.4.6 Infrastructure Node Network Connections

Like compute nodes, infrastructure nodes for *Compact Cloud* are each equipped with three NICs. The first NIC is connected to the Admin/PXE network in untagged mode. The second and third NICs are bonded (using Linux bonding) into a single logical interface (LACP mode) that serves Management, SAN, Storage Replication, Public, and Private networks on tagged-mode VLANs.

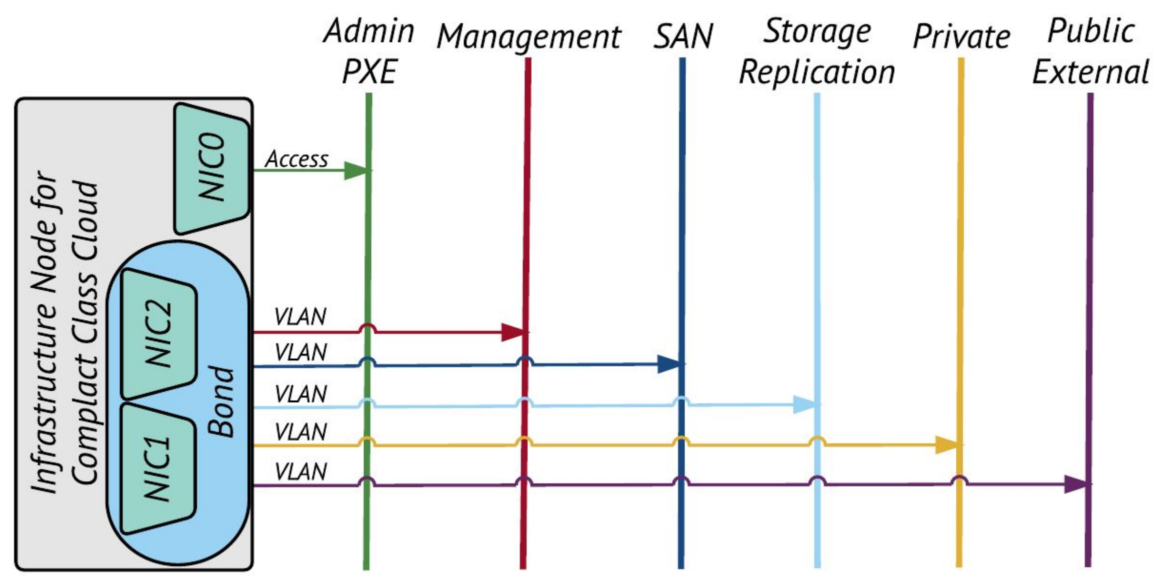


Figure 8. Compact Cloud - infrastructure node to network interconnection

4.5 Cloud Networking

4.5.1 Controller Node Networking

Inside *Compact Cloud* controller VMs, the Admin/PXE, Management, and SAN networks are terminated on individual vNICs. A fourth vNIC serves the VxLAN tunnels used for tenant networks, and a fifth vNIC is controlled by Open vSwitch and serves the Public network.

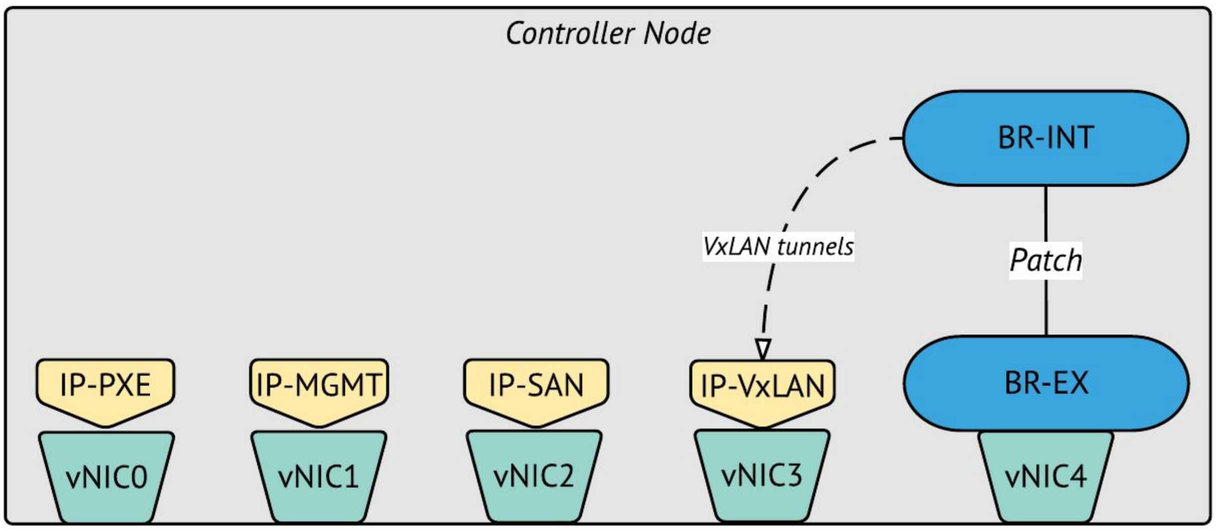


Figure 9. Compact Cloud - controller networking

4.5.2 Compute Node Networking

Inside *Compact Cloud* compute nodes the Admin/PXE network sits on the 1GbE interface. Management and SAN networks are carried over the bonded interface in VLANs. Other networks are served by OpenVSwitch, which is under Neutron control.

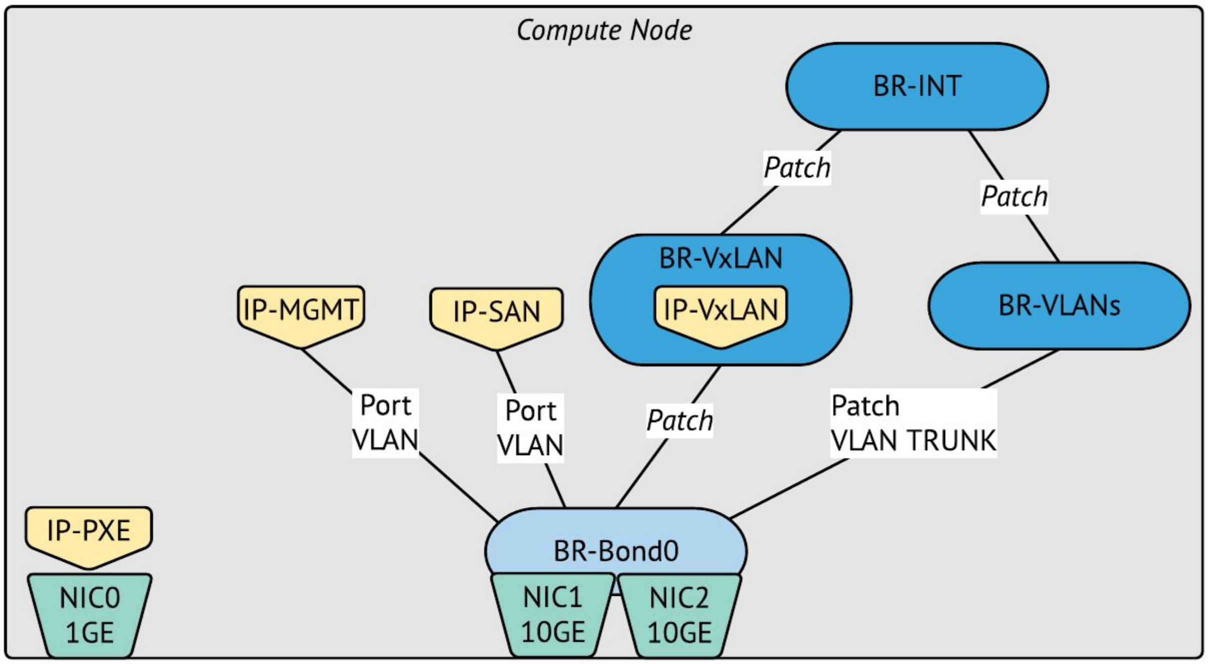


Figure 10. Compact Cloud - compute node networking

4.5.3 Storage Controller VM Networking

In storage controller VMs, all networks are connected to individual vNICs.

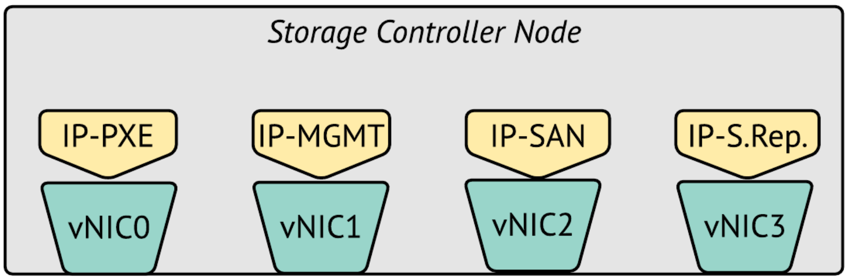


Figure 11. Compact Cloud - storage controller nodes

4.5.4 Ceph OSD Node Networking

On Ceph OSD nodes, the Admin/PXE network sits on the 1GbE interface. Other networks are served as VLANs extracted from the two bonded interfaces.

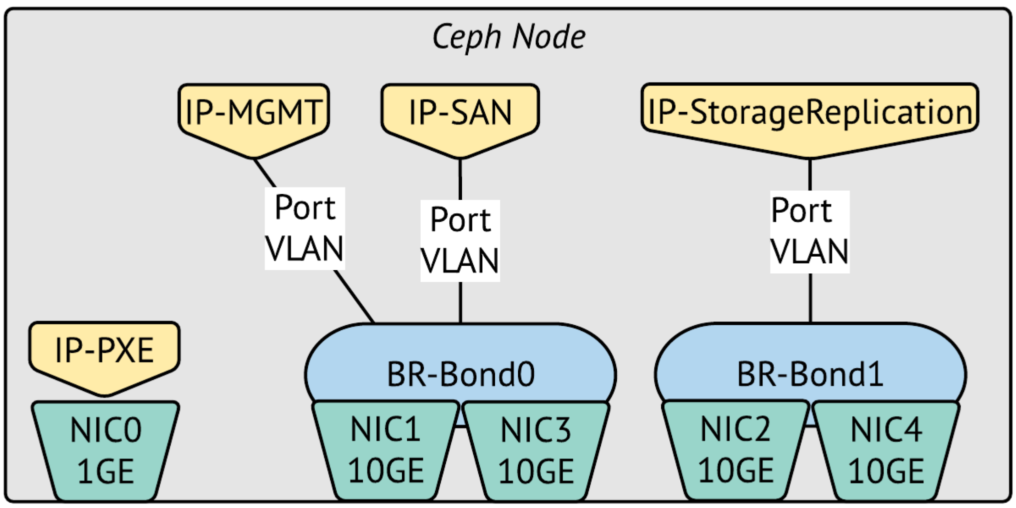


Figure 12. Compact Cloud - Ceph OSD node networking

4.5.5 Monitoring and Database Node Networking

In *Compact Cloud* monitoring and MongoDB nodes, networks are directly connected to vNICs without use of LACP or VLANs.

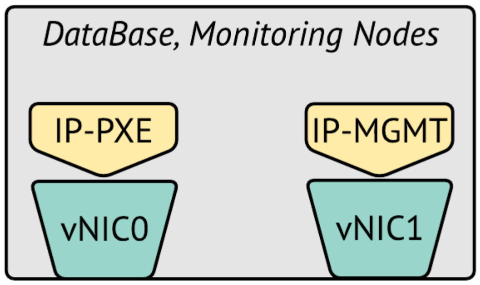


Figure 13. Compact Cloud - monitoring node networking

4.5.6 Infrastructure Node Networking

All network segments connecting to each *Compact Cloud* infrastructure node are bridged (standard Linux bridging). The Admin/PXE segment is separated on its own 1GbE interface. Two 10GbE interfaces, bonded in an LACP group, carry the remaining MOS network segments within VLANs.

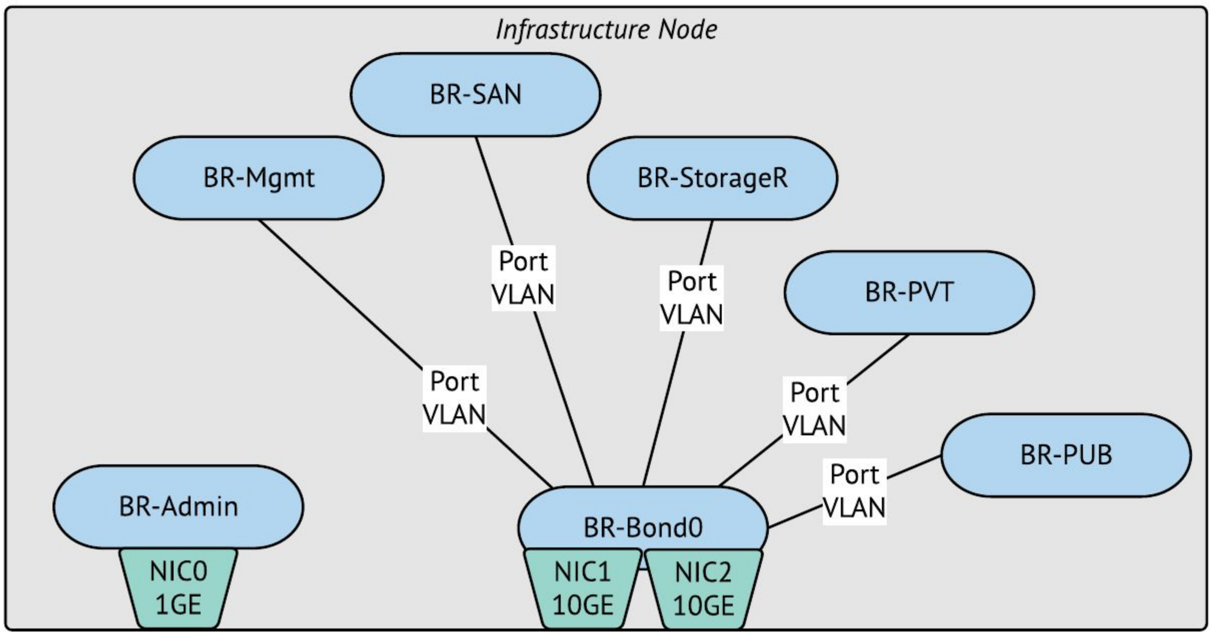


Figure 14. Compact Cloud - infrastructure node networking

4.6 Node Role VM Distribution Across Infrastructure Nodes

The *Compact Cloud* architecture deploys control plane node roles onto KVM virtual machines hosted on infrastructure nodes. Two configurations — minimal and recommended — are offered.

4.6.1 Minimal Footprint

Infrastructure nodes can be configured with a minimal footprint, providing controller and storage node redundancy but only including a single instance of the OpenStack and monitoring databases. While these databases can be protected (e.g., with mirroring), the minimal footprint does not offer true high availability — cloud operations or data collection will stall if issues occur within the relevant software, VMs, or physical hardware.

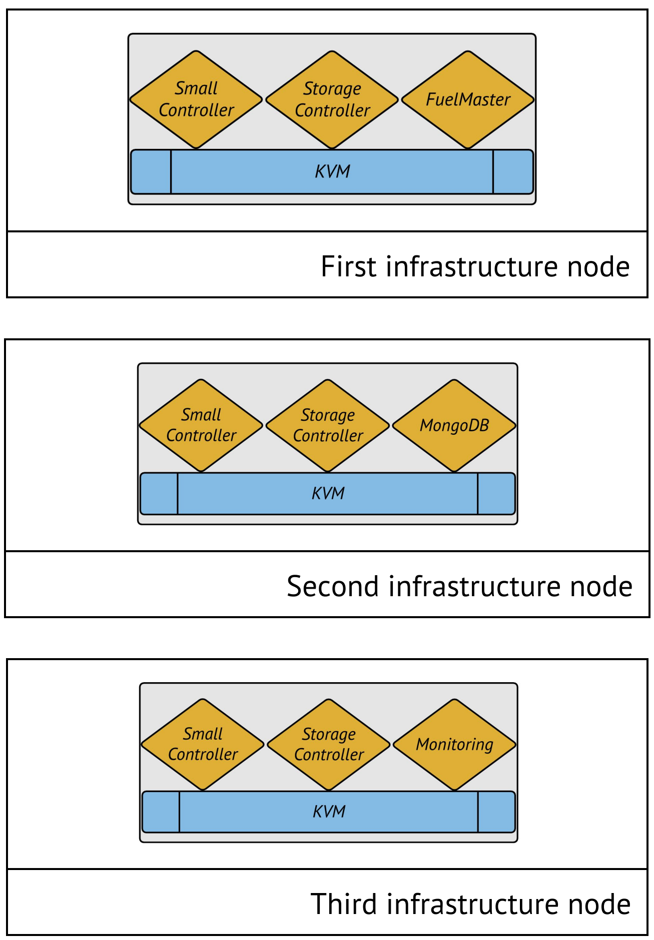


Figure 15. Compact Cloud - Node layout, minimal footprint.

4.6.2 Recommended Footprint

The *Compact Cloud* recommended footprint distributes database instances and uses clustering to achieve true HA.

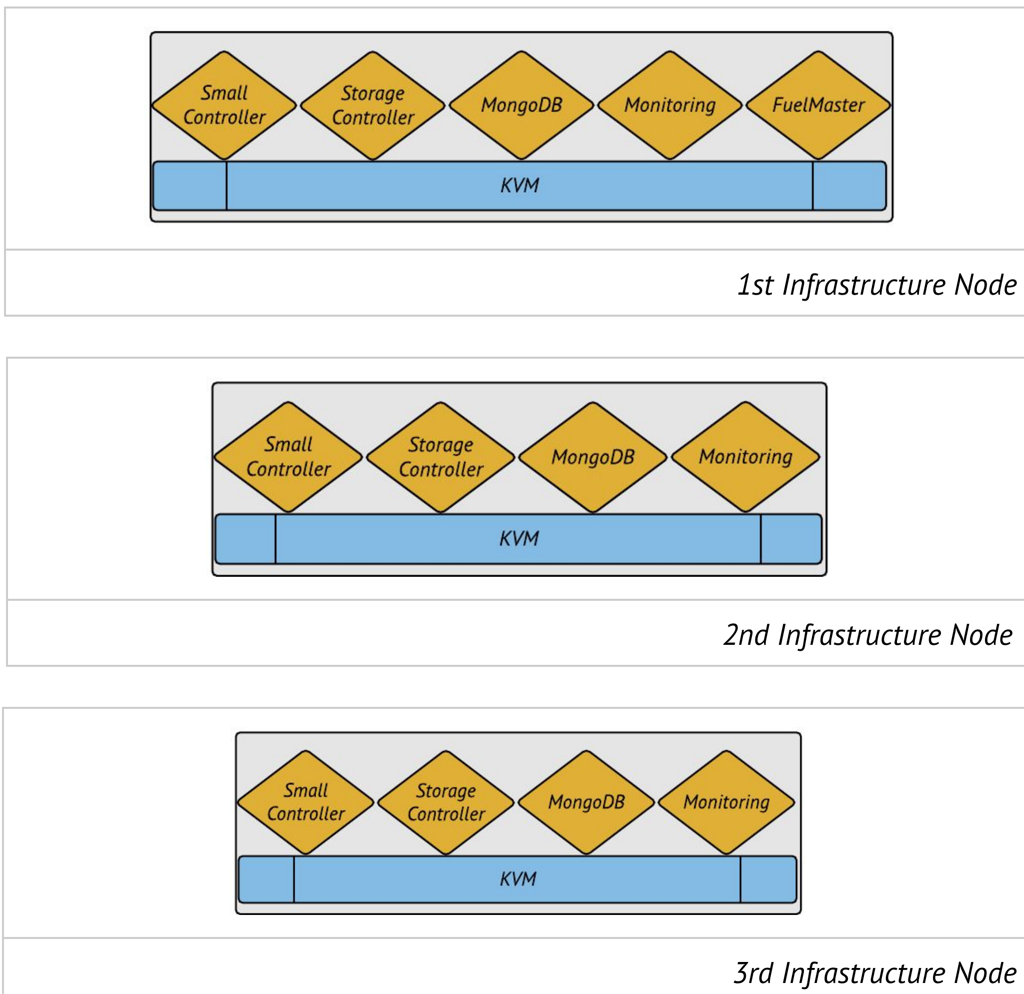


Figure 16. Compact Cloud - Node layout, recommended footprint.

4.7 Recommended Hardware and VM Configuration

The tables below summarize Mirantis and Dell EMC recommendations for configuring hardware and VMs to host node roles. Recommendations are discussed in greater detail, thereafter.

Table 2. Compact Cloud - Recommended VM configuration

Role	vCPUs	RAM	vHDDs	vNICs
Compact Controller	4	64GB	500GB	5
Storage Controller	2	32GB	100GB	5
MongoDB node	6	64GB	1.6TB	2
Monitoring node	6	64GB	1.6TB	2
Fuel Master	1	32GB	300GB	2

Table 3. Compact Cloud - Recommended hardware configuration

Role	Server model	CPU	RAM	Disks	NICs
Infrastructure node	Dell EMC PowerEdge R630	2x Intel® Xeon® E5-2650v4 (12 cores per CPU)	256GB	2x 1.2TB Intel SSD DC S3710 Series 4x 1.6TB Intel SSD DC S3610 Series	Intel® Ethernet Network Daughter Card X520-DA2 /1350-T2
Compute node	Dell EMC PowerEdge R630	2x Intel® Xeon® E5-2650v4 (12 cores per CPU)	256GB	2x 480GB Intel SSD DC S3610 Series	Intel® Ethernet Network Daughter Card X520-DA2 /1350-T2

Table 3. Compact Cloud - Recommended hardware configuration - Continued					
Ceph OSD node	Dell EMC PowerEdge R730xd	2x Intel® Xeon® E5-2650v4 (12 cores per CPU)	256GB	2x 200GB SAS (FlexBay) for OS 6x 1.6TB Intel SSD DC S3610 Series for Block Storage 15x 1.2TB SAS for Object Storage 3x 200GB Intel SSD DC S3710 Series	Intel® Ethernet Network Daughter Card X520-DA2 /1350-T2 Intel® Ethernet Converged Network Adapter X520-DA2 SFP+

4.7.1 Ceph OSD Nodes

Based on capacity and performance characteristics Mirantis and Dell EMC recommend Dell EMC PowerEdge R730xd to be used as a Ceph OSD node to store cloud data.

Configuration of Ceph OSD nodes primarily reflects required object storage capacity and block storage bandwidth requirements for the storage subsystem. Serially-attached storage (SAS) — relatively slow, but capacious and inexpensive — is often recommended for object storage. Performance can be increased cost-effectively by using SAS in conjunction with small SSDs, used to store the OSD journal[1], increasing the speed at which Journal data can be written.

In contrast, block storage requires high IOPS, making SSDs the only option in most cases. Intel SSD DC S3610 Series fit perfectly into that requirements.

[1] More info about OSD journal may be found here: http://docs.ceph.com/docs/giant/rados/configuration/journal-ref/

To address both object and block storage requirements appropriately, balancing cost and performance considerations and permitting fully-independent scaling for each storage type, Mirantis recommends using separate object and block storage hosts, configured as shown below:

- Object Storage server configuration
    - Dell EMC PowerEdge R730xd
    - 2x 300GB SATA disks for an operating system
    - 20x 2TB SAS disks to serve as Ceph OSDs
    - 4x 200GB Intel SSD DC S3710 Series for Ceph Journaling
    - 2x Intel® Xeon® E5-2630v4 (6 cores per CPU)
    - 96GB RAM
- Block Storage server configuration
    - Dell EMC PowerEdge R630
    - 2x 300GB SATA disks for an operating system
    - 6x 1.6TB Intel SSD DC S3610 Series to serve as Ceph OSDs
    - 2x Intel® Xeon® E5-2630v4 (10 cores per CPU)
    - 128GB RAM

Alternatively, clouds requiring both types of storage but requiring a smaller footprint can use the hybrid storage server configuration below, which maintains the same level of resiliency:

- Hybrid Object and Block Storage server configuration
    - Dell EMC PowerEdge R730xd
    - 2x 300GB SATA disks for an operating system
    - 15x 2TB SAS disks to serve as low speed Ceph OSDs
    - 3x 200GB Intel SSD DC S3710 Series for Ceph Journaling for low speed OSDs
    - 6x 1.6TB Intel SSD DC S3610 Series to serve as high speed Ceph OSDs
    - 2x Intel® Xeon® E5-2660v4 (14 cores per CPU)
    - 256GB RAM

4.7.2 MongoDB Node

Requirements for MongoDB depend on Ceilometer settings, such as the number of entities in a cloud we wish to monitor, what metrics we wish to collect from each entity, polling intervals, how many days we wish to store collected data, how many alarms we wish to define on top of collected data, and the evaluation interval for those alarms. Entities may be VMs, networks, objects in an object store, etc.

Based on Mirantis' experience with MongoDB v2.6, a basic configuration might provide a 1.6TB SSD, which is enough to store 30 days worth of data for up to 1500 entities with 25 metrics each, at a 60 sec polling interval. This basic configuration can live on a single VM in the controller set (small footprint), or (optimal footprint) distributed across the set of three (3) controller nodes (infrastructure nodes).

For configurations that push this ceiling (i.e., more entities x metrics-per-entity, shorter polling interval, more alarms, shorter alarm-evaluation interval, etc.) we recommend using a dedicated physical server to host MongoDB. Shown below are storage requirements for a range of example configurations:

- 3000 entities with 25 metrics per entity, 100 alarms with 30 sec evaluation interval
  - Minimal polling interval 35 sec
  - 5.5TB is needed to keep the data for 30 days
- 3000 entities with 15 metrics per entity, 100 alarms with 30 sec evaluation interval
  - Minimal polling interval 19 sec
  - 6TB is needed to keep the data for 30 days
- 3000 entities with 10 metrics per entity, 100 alarms with 30 sec evaluation interval
  - Minimal polling interval 13 sec
  - 5.8TB is needed to keep the data for 30 days
- 3000 entities with 8 metrics per entity, 100 alarms with 30 sec evaluation interval
  - Minimal polling interval 10 sec
  - 6TB is needed to keep the data for 30 days
- 4500 entities with 10 metrics per entity, 100 alarms with 30 sec evaluation interval
  - Minimal polling interval 19 sec
  - 6TB is needed to keep the data for 30 days
- 4500 entities with 25 metrics per entity, 100 alarms with 30 sec evaluation interval
  - Minimal polling interval 48 sec
  - 6TB is needed to keep the data for 30 days
- 1500 entities with 25 metrics per entity, 100 alarms with 30 sec evaluation interval
  - Minimal polling interval 16 sec
  - 6TB is needed to keep the data for 30 days

- 1500 entities with 10 metrics per entity, 100 alarms with 30 sec evaluation interval
    - Minimal polling interval 7 sec
    - 6TB is needed to keep the data for 30 days

4.7.3 Monitoring Node

The configuration specified in Table 2. Compact Cloud - Recommended VM configuration will suffice for clouds up to 100 Compute nodes, where logs and time-series data are retained for up to 30 days. Actual data stored depends greatly on workload characteristics and numbers, however, and may vary widely — in some cases possibly even exceeding the capacity of the recommended configuration.

4.7.4 Infrastructure Node

The configuration specified in Table 3. Compact Cloud - Recommended hardware configuration incorporates requirements for all virtual nodes that will be collocated on Infrastructure nodes. A disk configuration consists of 3 pairs of SSDs (RAID 1):

- 1.2TB RAID1 for a host operating system and Controller/Fuel virtual disks
- 1.6TB RAID1 for a MongoDB virtual disk
- 1.6TB RAID1 for a Monitoring virtual disk

For a minimal footprint (no HA for MongoDB and Monitoring nodes), the MongoDB database or Monitoring virtual disk can be stored on a single 1.6TB RAID1 device. In this case, the physical node requires two Intel® Xeon® E5-2620v4 (8 cores per CPU).

4.7.5 Hardware Summary - Servers

Dell EMC and Mirantis recommend the following hardware on which the *Compact Cloud* reference architecture was developed and validated.

Dell EMC PowerEdge R630

The PowerEdge R630 two-socket rack server delivers uncompromising density and productivity. Part of the 13th generation of PowerEdge servers, the R630 is ideal for virtualization. The processor and memory density with up to 24 DIMMs of DDR4 RAM provides great memory bandwidth.

Dell EMC PowerEdge R730xd

The incredible versatility of the PowerEdge R730xd server delivers outstanding functionality in just 2U of rack space. With the Intel® Xeon® processor E5-2600 v4 product family and up to 24 DIMMs of DDR4 RAM, the R730xd has the processing cycles and threads necessary to deliver more, larger and higher-performing storage for virtual machines. Highly scalable storage, with up to sixteen 12Gb SAS drives and the high-performance Dell PowerEdge RAID Controller H730, can greatly accelerate data access for your virtualized environment.

Dell Networking

Compact Cloud uses the Dell Networking S3048-ON top-of-rack open networking switch, a high density 1000BASE-T switch optimized for software-defined networking with ; and the Dell Networking S4048-ON switch, which offers a similar architecture but provides 48 10GbE front-side ports, and 6 40GbE backplane ports per 1U rack unit.

Table 4. Compact Cloud - Switches

Switch Model	Quantity
Dell Networking S3048-ON	1
Dell Networking S4048-ON	2

4.7.6 Intel Solid State Storage

Performance of the *Compact Cloud* architecture is optimized in part by allocating storage to a range of devices whose performance characteristics meet different operational demands. These include:

- Relatively small capacity, high-speed Intel® S3710 solid state disks, optimized for write performance - used to store Ceph OSD journals
- Larger capacity, Intel® S3610 solid state disks, optimized for balanced read/write performance - used for local block storage (volume and ephemeral storage) on compute nodes

Plus serial-attached (SAS) storage for objects.

4.8 Cloud Limits

This table summarizes tested upper limits for capacity and loading, and upper bounds for time required to perform critical operations (e.g., recovery from several failure modes, time to spawn a new workload, etc.).

Table 5. Compact Cloud - Capacity/utilization upper limits

Parameter	Upper limit
Number of hypervisors under one control plane	50
Number of storage nodes under one control plane	27
Number of simultaneously running workloads	3050
Number of workloads being run at once	300
Number of registered tenants	595
Number of registered users	29600
Number of users working with cloud simultaneously	3100
Percentage of vCPUs utilisation	98%
Percentage of RAM utilisation	98%
Percentage of Network (all kind of) bandwidth utilisation	99%
Percentage of Storage (all kind of) utilization	89%
Time to recover after failure that leads to permanent disruption of service	0'11"
Time to recover after failure that leads to failure of new requests	0'19"
Time to recover after failure that leads to failure of any request (new or currently executing)	7'24"
Time to spawn/terminate one workload	0'11"
Time to spawn/terminate the maximum "Number of workloads being run at once" (see above)	5'

5 Deployment Guide

In this section, we describe how to deploy *Compact Cloud* with MOS 9.1 on Dell EMC hardware. For more information, please see the Mirantis OpenStack documentation.

5.1 Hardware Specification

Three types of hardware nodes are used: Infrastructure, Compute, and Storage.

Table 6. Compact Cloud - Hardware node configuration

Node Type	Specification	Amount
Infrastructure	Dell PowerEdge R630 with ● 2x CPU Intel Xeon E5-2650 v4 @ 2.20hz (12 cores) ● 256GB RAM ● 2x Intel s3610 1.6TB SSD ● 2x Intel s3710 400GB SSD	3
Compute	Dell PowerEdge R630 with ● 2x CPU Intel Xeon E5-2650 v4 @ 2.20hz (12 cores) ● 256GB RAM ● 1x Intel s7310 400GB SSD	4
Storage	Dell PowerEdge R730xd with ● 2x CPU Intel Xeon E5-2650 v4 @ 2.20hz (12 cores) ● 256GB RAM ● 6x Intel s3610 1.6TB SSD ● 3x Intel s3710 200GB SSD ● 15x SAS drive 1.2TB ● 2x SAS drive 1.2TB (Flexbay)	4

The Fuel Master is deployed on one of the Compute nodes and then moved to a virtual machine on one of Infrastructure nodes. The Compute node initially used by the Fuel Master is then added to the cluster as a regular Compute node.

Dell Networking switches used in the reference deployment are identified in Table 4. Compact Cloud - Switches.

5.2 Deployment Overview

The architecture, described in detail in sections 1-4 of this document, is hereunder summarized in terms of a practical deployment:

Table 7. Compact Cloud - Deployment specification

Component	PoC Configuration
Controller High Availability	HA Controller Configuration (per standard Mirantis Reference Architecture)
Glance Back-end	Ceph/RBD
Object Storage	Ceph/RadosGW
Cinder Back-end	Ceph/RBD
Keystone identity backend	● MySQL for default domain (services userIDs) ● LDAP for end-users (multiple domains)
Keystone assignment backend	MySQL
Host OS	Ubuntu 14.04
Networking	Neutron with tunneling segmentation
Number of Controllers	3 Nodes
Number of Compute Nodes	3 Nodes
Number of Storage Nodes	4 nodes Ceph OSD and 3 nodes Ceph Monitors
Other Nodes	3 Virtual nodes (Infrastructure nodes as per Reference Architecture)
Syslog Server	Fuel by default

Table 7. Compact Cloud - Deployment specification - Continued	
Monitoring	3 StackLight: Infrastructure Alerting, Elasticsearch Kibana, InfluxDB Grafana nodes
Release	Mirantis OpenStack 9.1 (Mitaka)
Ceilometer	3 MongoDB nodes
Related Projects	N/A

5.2.1 Fuel Plugin & Component Overview

Fuel plugins and components used in deploying *Compact Cloud* are versioned below.

Table 8. Compact Cloud - Fuel plugins and components required

Plugin/Component name	Plugin | package version
elasticsearch_kibana	*0.10.2 | 4.0.0*
lma_infrastructure_alerting	*0.10.2 | 4.0.0*
influxdb_grafana	*0.10.2 | 4.0.0*
lma_collector	*0.10.2 | 4.0.0*
standalone-ceph	*2.0.0 | 4.0.0*

5.2.2 Additional Extensions and Integrations

Additional integrations, not provided by the above Fuel plugins, are enumerated here.

Table 9. Compact Cloud - Additional integrations required

Extension	Component
Template for network configuration	*Fuel*
Post-install Ceph configuration rearrangement	*Ceph*

5.3 Network Layout

Details specific to *Compact Cloud* deployed network configuration are summarized below.

Table 10. Compact Cloud - Network configuration detail

Network name	Speed	Port mode	IP Range	VLAN	Interface
IPMI network	1 Gbps	Untagged	172.18.232.0/24	100	IPMI/Mgmt
Admin/PXE network	1 Gbps	Untagged	10.20.0.0/24	120	br-admin
Management network	10 Gbps	Tagged	192.168.0.0/24	140	br-mgmt
SAN	10 Gbps	Tagged	192.168.3.0/24	190	br-san
Storage network	10 Gbps	Tagged	192.168.1.0/24	180	br-storage
Public network	10 Gbps	Tagged	172.16.224.0/24	160	br-public
Private network	10 Gbps	Tagged	192.168.2.0/24	200	br-private

5.4 Fuel Master Node Installation

Compact Cloud is deployed using the Fuel deployer. A generalized, step-by-step guide for Fuel Master Node installation can be found in the Mirantis OpenStack documentation.

IMPORTANT: MOS 9.1, used in this deployment, is available as an upgrade to MOS 9.0. Please download the MOS 9.0 ISO, then create the Fuel Master Node according to the instructions linked above. If internet access is available during installation, the upgrade to MOS 9.1 will proceed automatically.

This table summarizes settings we made during deployment of the *Compact Cloud* PoC on Dell EMC hardware. These are in most ways customizable:

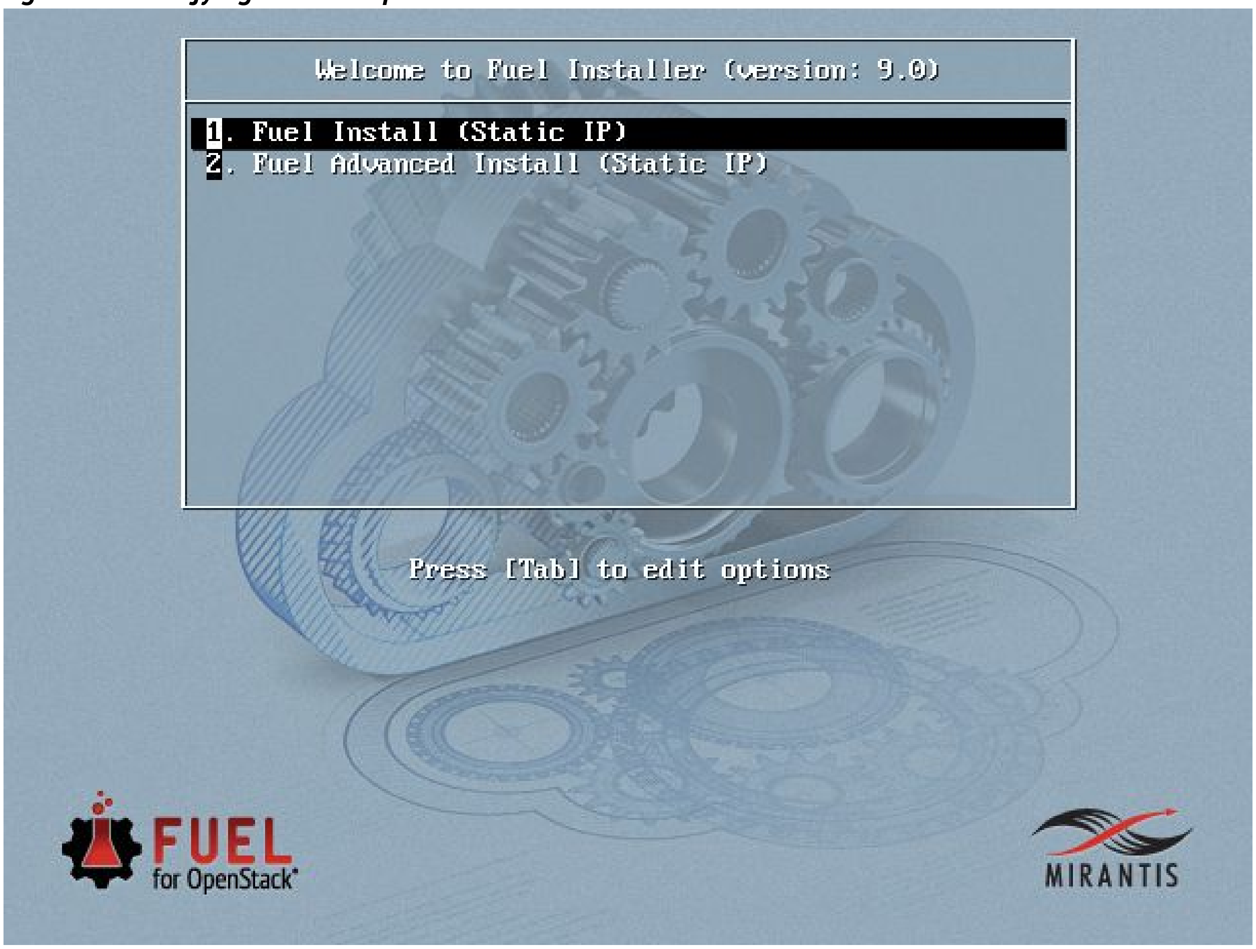
Table 11. Compact Cloud - Settings used in deployment

Hostname	fuel.domain.tld
Enabled Interface	eth0, eth1
Interface for PXE	eth0
DHCP IP address	10.20.0.2/24
DHCP Pool range	10.20.0.3 - 10.20.0.254
Management Interface	eth1
Management IP Address	172.16.224.4/24
Gateway	172.16.224.1
DNS server	8.8.8.8
Domain	domain.tld
Search domain	domain.tld
Feature Groups enabled	"Advanced Features"

IMPORTANT: In our PoC, we used one of the Compute nodes, initially, as the Fuel node. Post deployment, we converted the Fuel Master into a virtual machine and migrated this VM to one of our infrastructure nodes, re-dedicating the server it previously occupied to the cluster as a normal Compute node. If you intend to do this, it is important to remember *not to change the name of the Fuel Master node in the process.*

Boot a server using the MOS 9.0 ISO. When the Fuel Installer menu appears, press the Tab key to modify boot parameters:

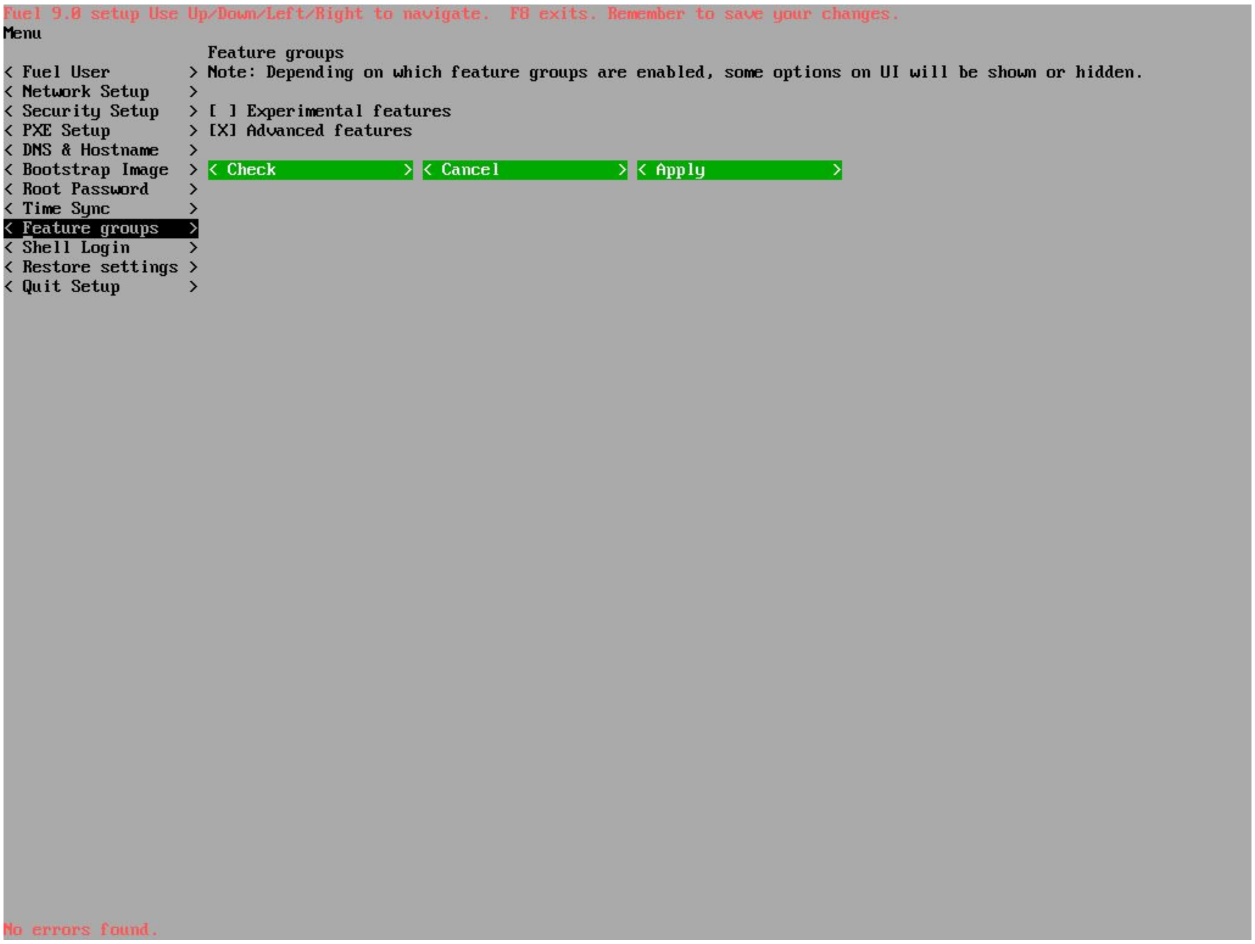
Figure 17. Modifying Fuel boot parameters



Make sure that the "showmenu" parameter is set to "yes," then hit the Enter key to proceed with installation. Wait until the OS is installed, the node is rebooted, and the Fuel Menu appears.

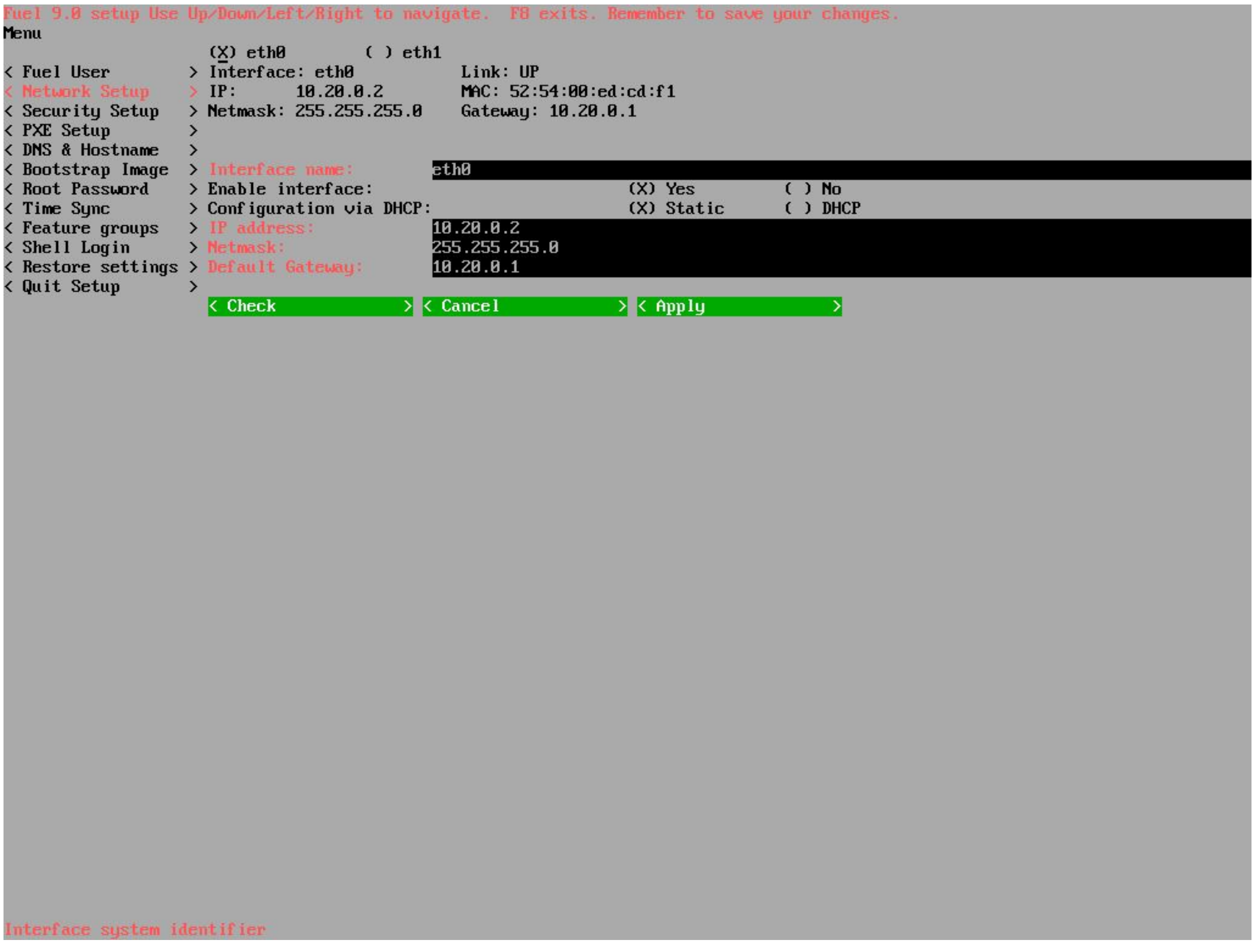
When the Fuel menu appears, navigate to the Feature Groups section and enable the "Advanced features" option:

Figure 18. Enabling Fuel Advanced Features



Change Network, PXE or DNS settings if needed (if, for example, the Fuel node has access to the internet over the second, rather than the first network interface).

Figure 19. Changing Fuel network, boot and other settings



Proceed with installation as described in the Fuel Installation Guide.

5.5 Post Installation Customization

Check that the MOS 9.1 upgrade has been applied. To do this, run "yum info fuel|grep repo" on the Fuel node's command line.

```
[root@fuel ~]# yum info fuel|grep repo
From repo    : mos9.0-updates
```

The repo should be "mos9.0-updates". If it isn't, follow the update instructions.

Next, install the Fuel plugins required for the deployment:

```
# fuel plugins --install elasticsearch_kibana-0.10-0.10.2-1.noarch.rpm
# fuel plugins --install influxdb_grafana-0.10-0.10.2-1.noarch.rpm
# fuel plugins --install lma_collector-0.10-0.10.2-1.noarch.rpm
# fuel plugins --install lma_infrastructure_alerting-0.10-0.10.2-1.noarch.rpm
# fuel plugins --install ldap-3.0-3.0.0-1.noarch.rpm
# fuel plugins --install standalone-ceph-2.0-2.0.0-1.noarch.rpm
# fuel plugins
id | name                        | version | package_version | releases
---+-----------------------------+---------+-----------------+--------------------------------------------
1  | elasticsearch_kibana        | 0.10.2  | 4.0.0           | ubuntu (liberty-8.0, liberty-9.0, mitaka-9.0)
2  | influxdb_grafana            | 0.10.2  | 4.0.0           | ubuntu (liberty-8.0, liberty-9.0, mitaka-9.0)
3  | lma_collector               | 0.10.2  | 4.0.0           | ubuntu (liberty-8.0, liberty-9.0, mitaka-9.0)
4  | lma_infrastructure_alerting | 0.10.2  | 4.0.0           | ubuntu (liberty-8.0, liberty-9.0, mitaka-9.0)
5  | ldap                        | 3.0.0   | 3.0.0           | ubuntu (mitaka-9.0)
6  | standalone-ceph             | 2.0.0   | 4.0.0           | ubuntu (mitaka-9.0)
```

Because we will use the standalone-ceph plugin we need to add a definition of the new "san" bridge in the Libvirt VM template for "virt" nodes:

```
[root@fuel ~]# vim
/etc/puppet/mitaka-9.0/modules/osnailyfacter/templates/vm_libvirt.erb
<domain type='kvm'>
  <devices>
…
      <interface type='bridge'>
      <source bridge='br-mesh'/>
      <model type='virtio'/>
      </interface>
      <interface type='bridge'>
      <source bridge='br-san'/>
      <model type='virtio'/>
      </interface>
      <serial type='pty'>
...
  </devices>
</domain>
```

5.6 OpenStack Environment Deployment

Once the Fuel Master is complete, we can use it to configure and deploy the environment.

The following settings were used during installation of OpenStack:

Table 12. Compact Cloud - OpenStack basic environment settings

Name	*Compact Cloud*
OpenStack Release	Mitaka on Ubuntu 14.04
Compute	KVM
Networking Setup	Neutron with tunneling segmentation
Storage Backends	Yes, use Ceph
Additional services	Murano, Ceilometer (OpenStack Telemetry)

5.6.1 Network Settings

First, create a new network group for the SAN:

```
[root@fuel ~]# fuel2 network-group create san -N $GROUP_ID -C $NET_CIDR -V
$VLAN
```

The term $GROUP_ID can be found in output from the "fuel2 network-group list" command (it must be the same for all networks of your environment). $NET_CIDR is the CIDR range for the SAN. $VLAN represents the VLAN index for the SAN. For example:

```
[root@fuel ~]# fuel2 network-group create san -N 6 -C 192.168.3.0/24 -V 190
```

Then modify network settings on the Network Settings tab of the Fuel web UI.

Here is how network settings were configured for this test deployment:

Figure 20. Network settings

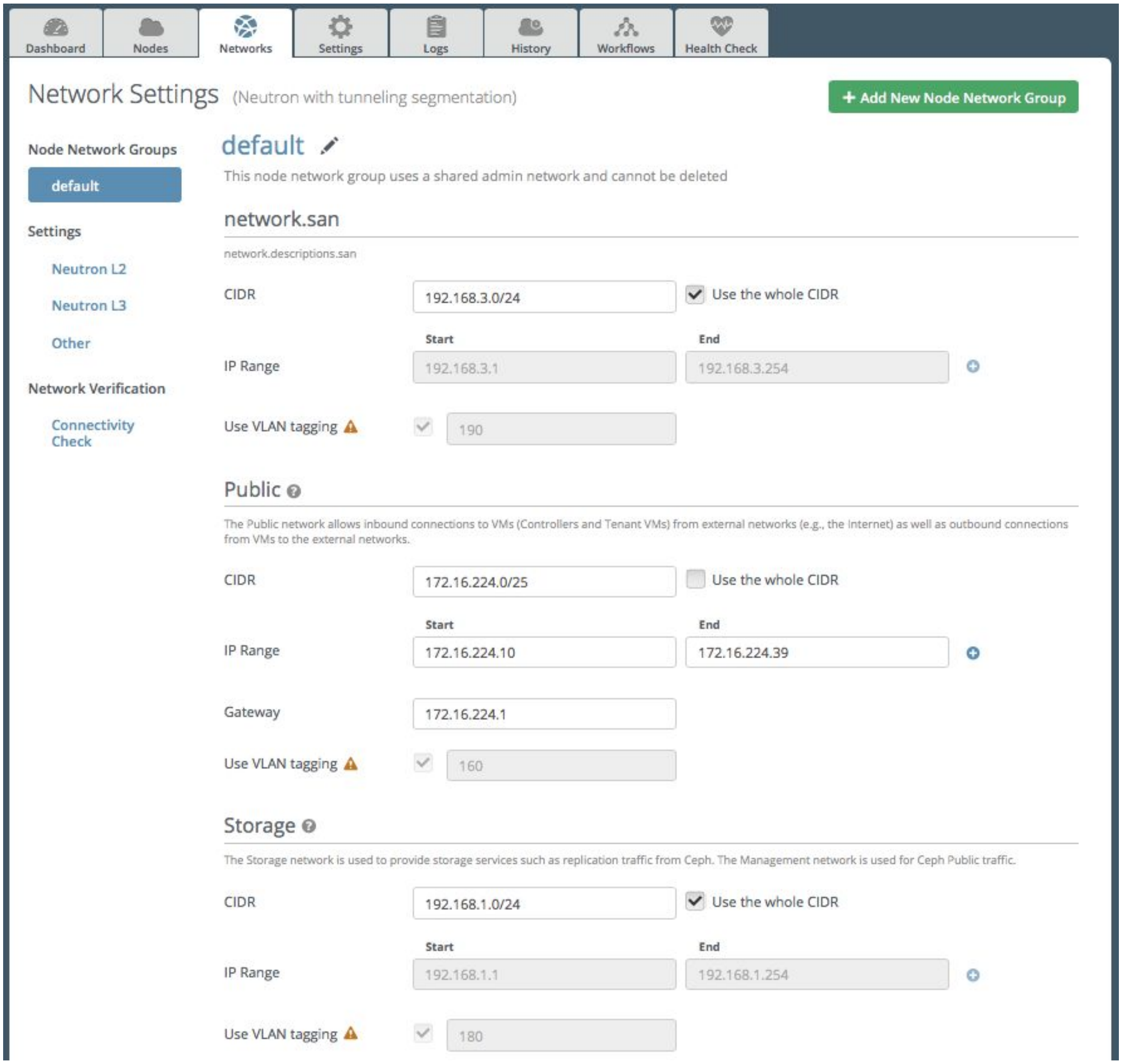


Figure 21. Network settings (continued 1)

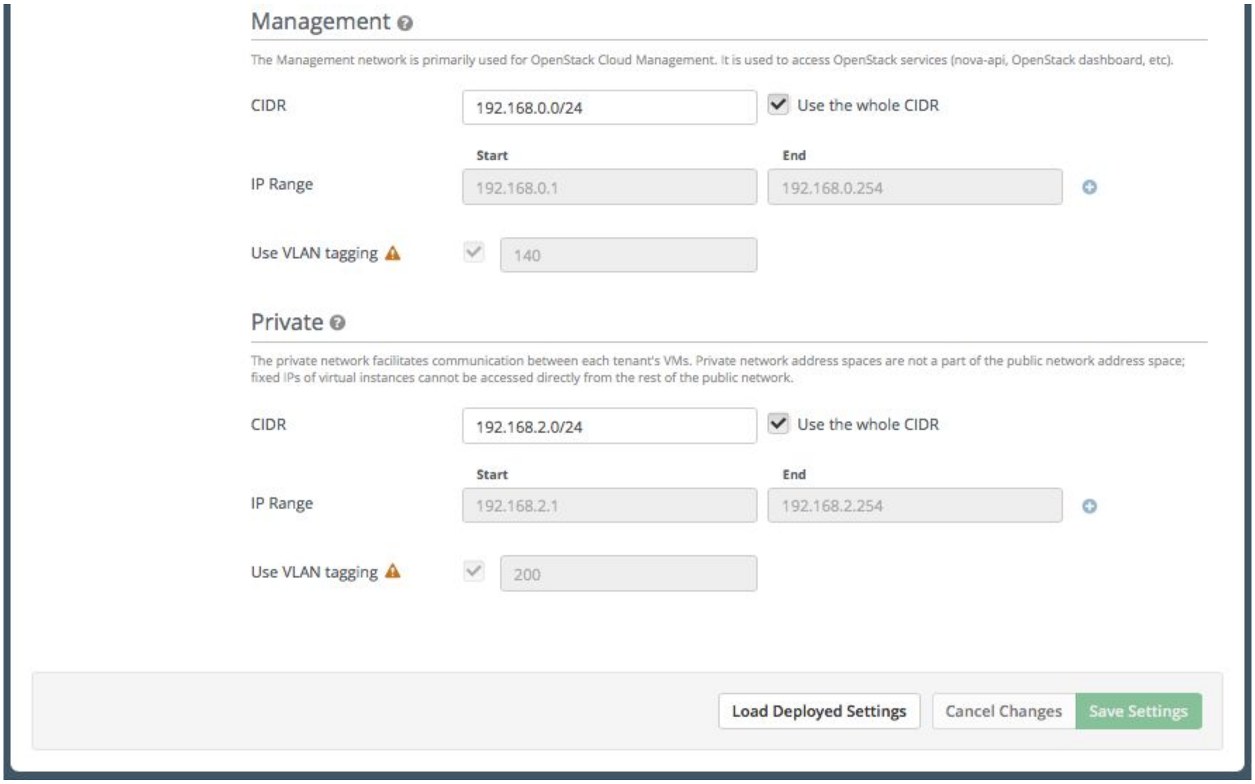


Figure 22. Network settings (continued 2)

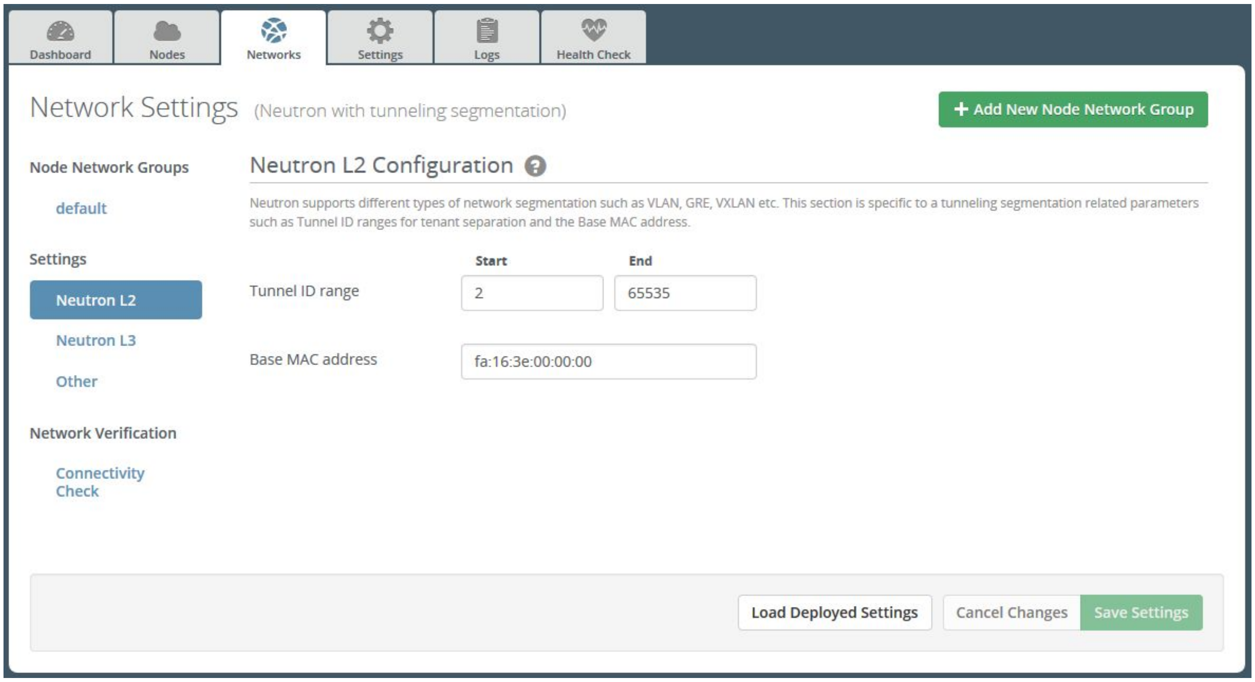


Figure 23. Network settings (continued 3)

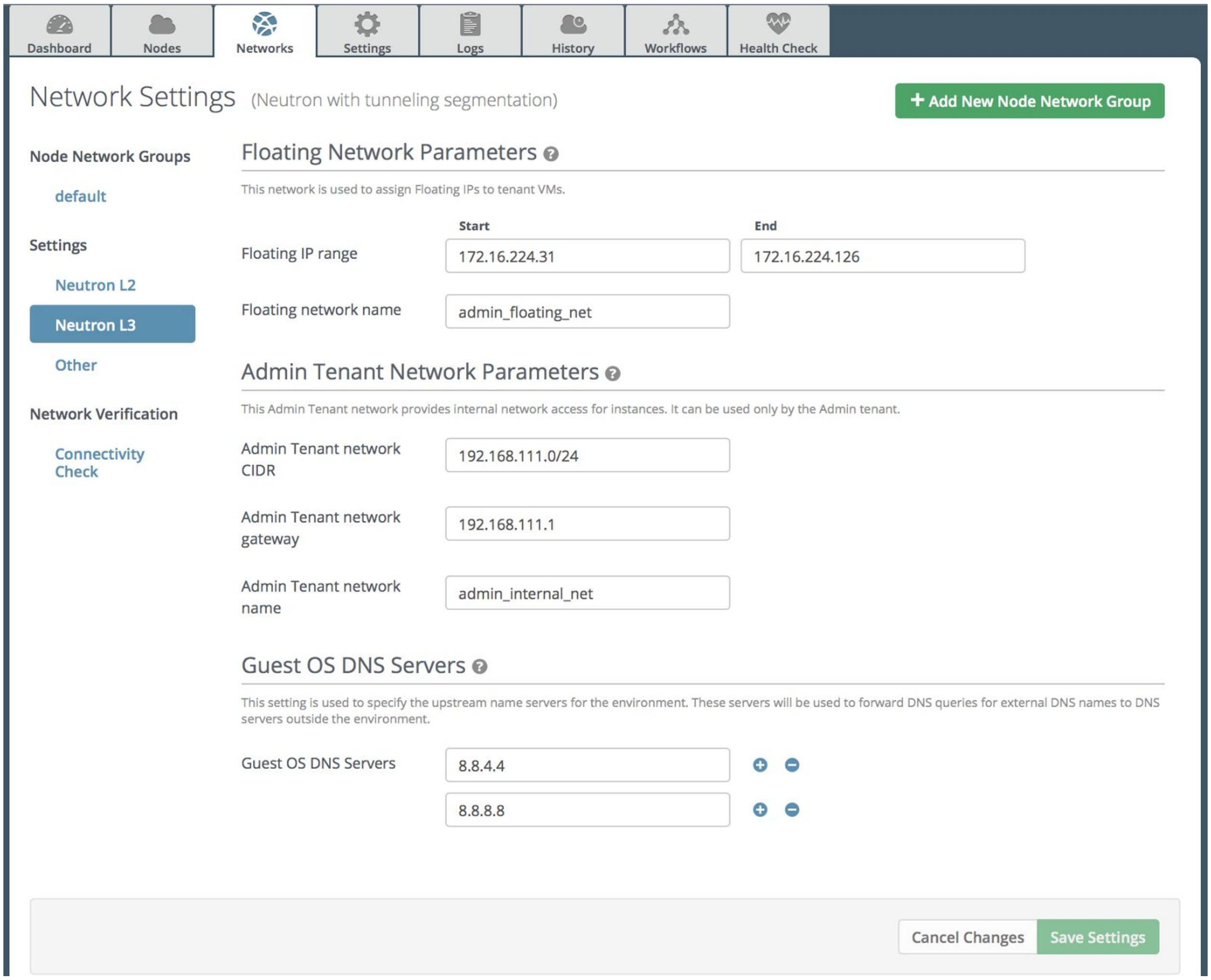
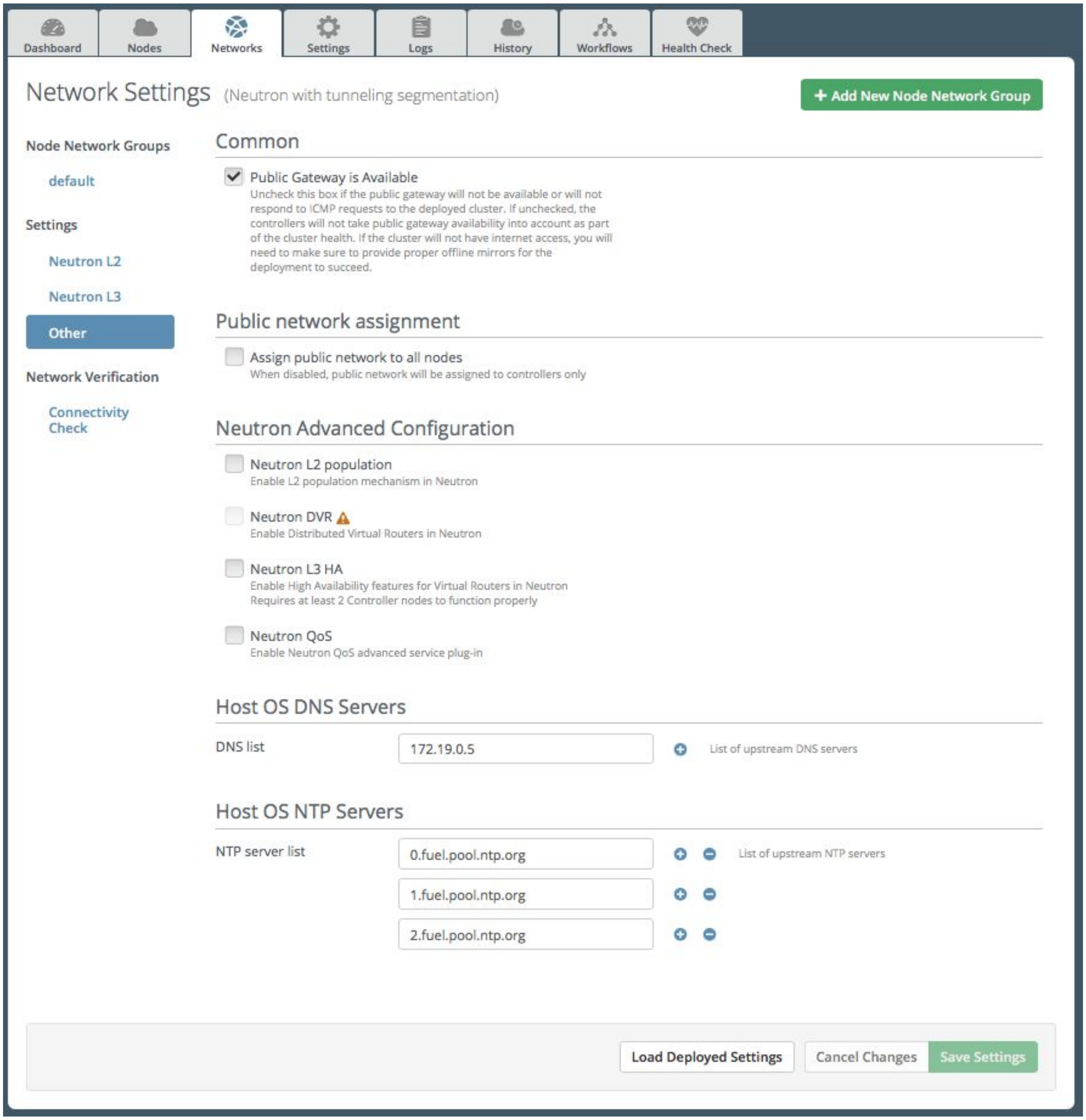


Figure 24. Network settings (continued 4)



5.6.2 Upload network template

Because we are using untagged networks on virtualized nodes and tagged networks on baremetal nodes, we use network templates to define different interface settings for each node type. See the network template example in the Appendices.

Upload the network template to the environment:

```
<fuel_master># fuel2 network-template upload -f $NET_TEMPLATE_FILE $ENV_ID
```

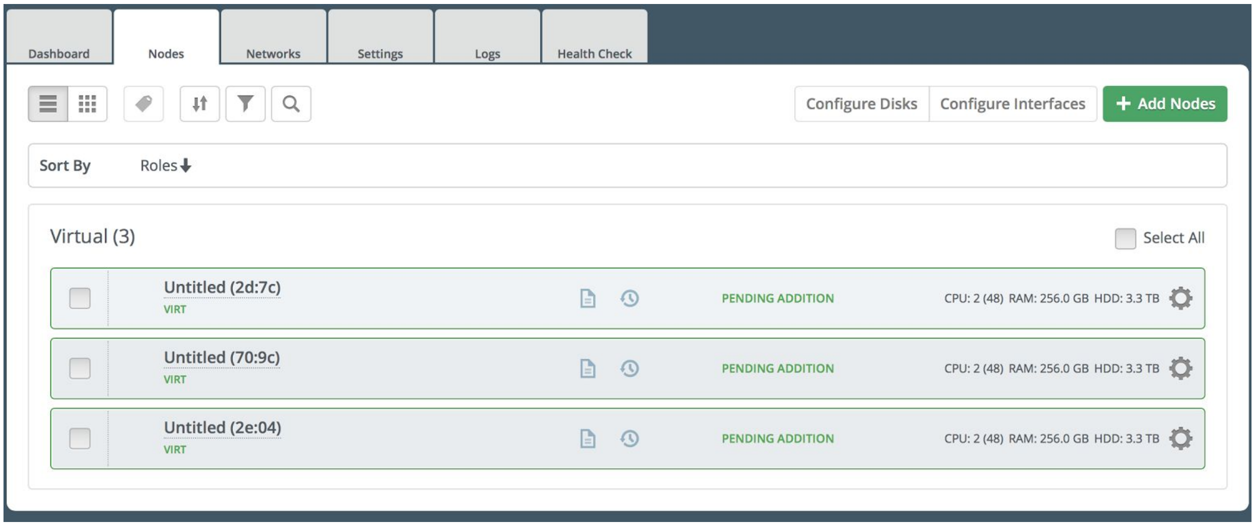
Where $NET_TEMPLATE_FILE represents the network template file without its extension, and $ENV_ID represents the environment ID. For example:

```
[root@fuel ~]# ls *network-template*
Dell-MOS9.0-CompactCloud-network-template.yaml
[root@fuel ~]# fuel2 network-template upload -f Dell-MOS9.0-CompactCloud-network-template 6
```

5.6.3 Add Infrastructure Nodes for Control Plane

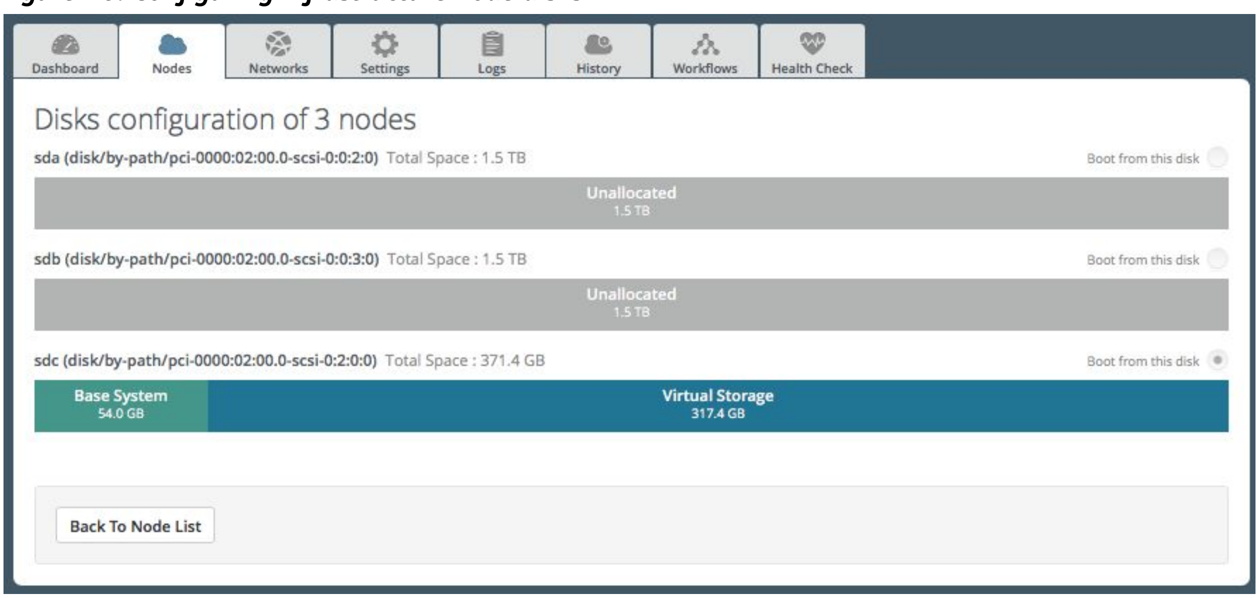
Add Infrastructure nodes to the environment and assign [the Virtual role] to them:

Figure 25. Adding infrastructure nodes with Virtual role



Select all three nodes and configure their disks as shown:

Figure 26. Configuring infrastructure node disks

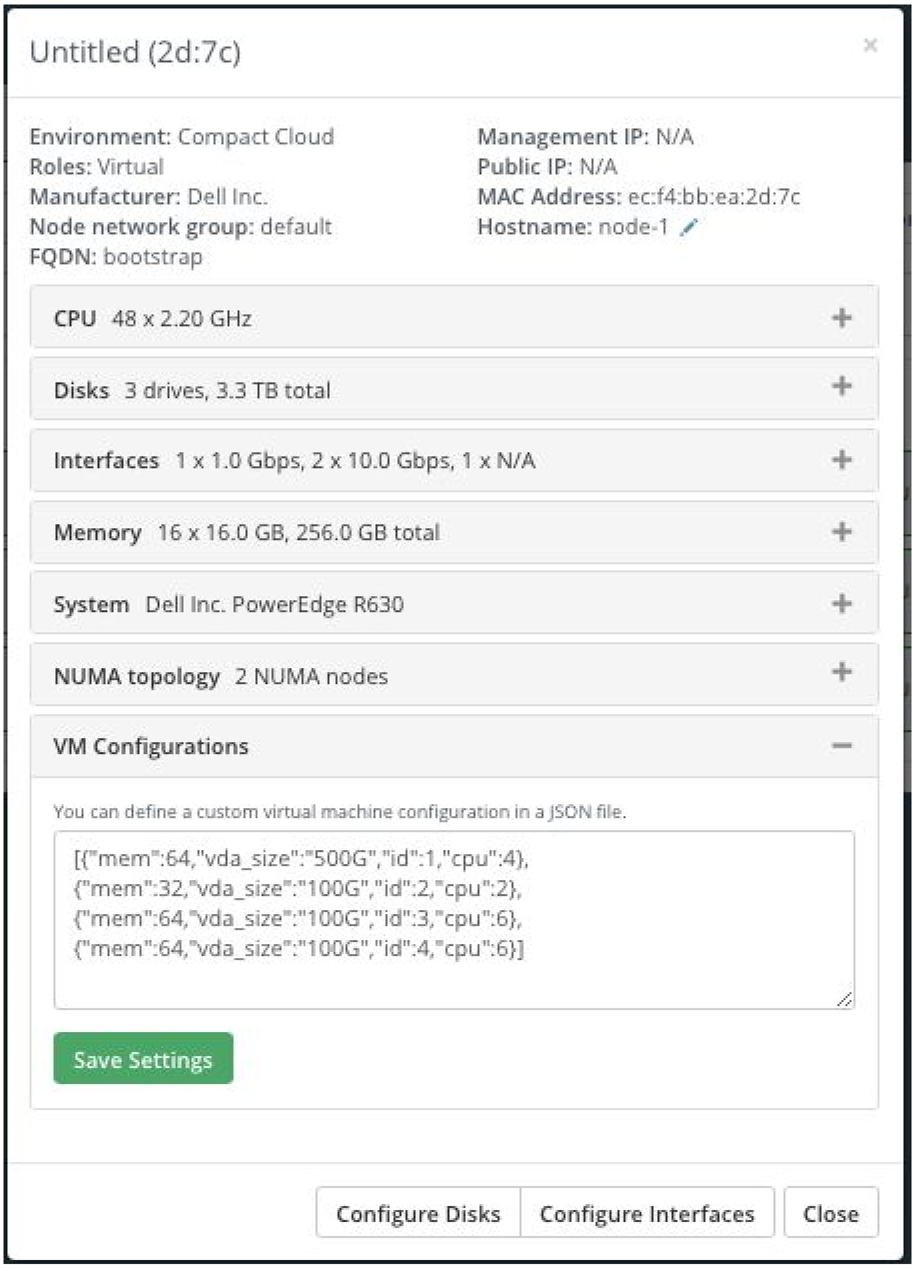


:

5.6.4 Creating VMs to Host Controller Components

For each node, add VM definitions in "VM Configurations" section for the VMs that will host the compact controller, storage controller, MongoDB, and monitoring as described in section 4.6:

Figure 27. Add VM configuration details



You can define the amount of memory in GB, the size of the disk, and the number of CPU cores for each VM. VM IDs must be unique within each hardware node on which VMs are provisioned. As a

general best practice, it makes sense to give each VM in an environment a globally unique ID, as this will prevent ID conflicts later, if migration of VMs is required.
For this exercise, we've defined 4 VMs on each Virt node, as described below, as shown in the illustration immediately preceding, and as discussed in section 4.6.2, above:

- Controller VMs with IDs 1, 5, 9
    - 500GB virtual disk, 64GB RAM, 4 vCPU
- Storage Controller VMs with IDs 2, 6, 10
    - 100GB virtual disk, 32GB RAM, 2 vCPU
- MongoDB VMs with IDs 3, 7, 11
    - 100GB virtual disk, 64GB RAM, 6 vCPU
- Monitoring VMs with IDs 4, 8, 12
    - 100GB virtual disk, 64GB RAM, 6 vCPU

NOTE: Do not set the "created" parameter to "true" as this instructs Fuel to assume a pre-existing VM (rather than creating a new VM) during provisioning.

VMs can also be created from the Fuel CLI:

```
<fuel_master># fuel2 node create-vms-conf <VIRT_NODE_ID> --conf '[{"mem":64,
"vda_size":"500G", "id":1, "cpu":4}, {"mem":32, "vda_size":"100G", "id":2,
"cpu":2}, {"mem":64, "vda_size":"100G", "id":3, "cpu":6}, {"mem":64,
"vda_size":"100G", "id":4, "cpu":6}]'
```

NOTE: To add a VM to the deployed environment you must use the CLI, and the command must be formatted as shown, with already-created VMs flagged as "created": true. You can cause new VMs to be created by Fuel by setting "created": false, as shown below:

```
<fuel_master># fuel2 node create-vms-conf <VIRT_NODE_ID> --conf '[{"mem":16,
"vda_size":"300G", "created":true, "id":1, "cpu": 8}, {"mem":4,
"vda_size":"300G", "created":true, "id":2, "cpu":2}, {"mem":4,
"vda_size":"300G", "created":false, "id":10, "cpu":2}]'
```

Or you can just skip the "created" parameter for VMs you need created, since "created": true is the default.

```
<fuel_master># fuel2 node create-vms-conf <VIRT_NODE_ID> --conf '[{"mem":16,
"vda_size":"300G", "created":true, "id":1, "cpu":8}, {"mem":4,
"vda_size":"300G", "created":true, "id":2, "cpu":2}, {"mem":4,
"vda_size":"300G", "id":10, "cpu":2}]'
```

After that, for a brand-new *Compact Cloud* environment, you can cause Fuel to provision VM(s) by clicking the "Provision VMs" button in the Fuel web UI. However, if you are creating additional VM(s) in an already deployed environment, you must use the CLI:

```
<fuel_master># fuel node --node <VIRT_NODE_ID> --deploy
```

5.6.5 Attach SSDs to MongoDB and Monitoring Nodes

Once VMs are provisioned and discovered, we need to replace their virtual disks with 1.5GB SSDs on the MongoDB and Monitoring nodes. To do this, perform the following operation on each Virt node:

- SSH to the Virt node #1
- Using the "virsh destroy <ID>_vm" command, stop the Monitoring and MongoDB VMs with IDs 3 and 4 (7, 8 at Virt node #2 and 11, 12 at Virt node #3)
- We'll address SSDs via their Device IDs. Determine the device IDs for each 1.5GB SSD by grepping a listing of the "/dev/disk/by-id/" directory:

```
root@node-1:~# ls -l /dev/disk/by-id/|grep wwn|grep sd[ab]
lrwxrwxrwx 1 root root  9 Nov 11 05:58 wwn-0x55cd2e404c341e15 -> ../../sdb
lrwxrwxrwx 1 root root  9 Nov 11 05:58 wwn-0x55cd2e404c341e31 -> ../../sda
```

- Use the "virsh edit <ID>_vm" command to modify the "disk" section. Assign each VM to a dedicated SSD.

```
    <disk type='block' device='disk'>
      <driver name='qemu' type='raw' cache='none'/>
      <source dev='/dev/disk/by-id/wwn-0x55cd2e404c341e15'/>
      <target dev='vda' bus='virtio'/>
```

- Start the VMs using the "virsh start <ID>_vm" command.
- Wait a couple of minutes to let nodes be discovered by Fuel. Verify in the Fuel UI that the disk size of the nodes has changed.

5.6.6 Assign Roles to Cluster Nodes

Add remaining hardware nodes to the cluster and assign required roles to the spawned virtual machines and hardware nodes using the Fuel web UI. Alternatively, you can do this with Fuel CLI by issuing the following command:

```
<fuel_master># fuel --env-id=<ENV_ID> node set
--node-id=<NODE1_ID>[,<NODE2_ID>] --role=controller
```

Here is an example of a full configuration:

```
[root@fuel ~]# fuel node
id | status    | name                | cluster | ip           | mac                 | roles
---+-----------+--------------------+---------+-------------+--------------------+--------------------------------------
-------------------------
19 | ready     | 12_vm               |       6 | 10.20.0.18 | 52:54:00:13:5d:94 | elasticsearch_kibana, influxdb_grafa
infrastructure_alerting
21 | ready     | 4_vm                |       6 | 10.20.0.24 | 52:54:00:88:d9:55 | elasticsearch_kibana, influxdb_grafa
infrastructure_alerting
11 | ready     | 3_vm                |       6 | 10.20.0.23 | 52:54:00:d3:2c:da | mongo
13 | ready     | 1_vm                |       6 | 10.20.0.21 | 52:54:00:16:86:1c | controller
 5 | ready     | compute-2           |       6 | 10.20.0.6  | ec:f4:bb:ea:6f:c4 | compute
 3 | ready     | infra-3             |       6 | 10.20.0.5  | ec:f4:bb:ea:2e:04 | virt
 2 | ready     | infra-2             |       6 | 10.20.0.4  | ec:f4:bb:ea:70:9c | virt
20 | ready     | 5_vm                |       6 | 10.20.0.13 | 52:54:00:dd:7f:f5 | controller
 1 | ready     | infra-1             |       6 | 10.20.0.3  | ec:f4:bb:ea:2d:7c | virt
12 | ready     | 9_vm                |       6 | 10.20.0.20 | 52:54:00:5c:ae:55 | controller
22 | ready     | 2_vm                |       6 | 10.20.0.22 | 52:54:00:13:41:45 | ceph-mon
17 | ready     | 7_vm                |       6 | 10.20.0.17 | 52:54:00:5e:2c:c1 | mongo
14 | ready     | 6_vm                |       6 | 10.20.0.14 | 52:54:00:b9:f5:84 | ceph-mon
15 | ready     | 10_vm               |       6 | 10.20.0.16 | 52:54:00:0e:40:bd | ceph-mon
16 | ready     | 8_vm                |       6 | 10.20.0.15 | 52:54:00:8d:14:47 | elasticsearch_kibana, influxdb_grafa
infrastructure_alerting
18 | ready     | 11_vm               |       6 | 10.20.0.19 | 52:54:00:e8:96:c6 | mongo
 7 | ready     | ceph-1              |       6 | 10.20.0.11 | ec:f4:bb:ea:2b:1c | ceph-osd
 6 | ready     | compute-3           |       6 | 10.20.0.7  | ec:f4:bb:ea:6f:7c | compute
10 | ready     | ceph-4              |       6 | 10.20.0.9  | 24:6e:96:1f:4a:e4 | ceph-osd
 4 | ready     | compute-1           |       6 | 10.20.0.8  | ec:f4:bb:ea:2e:44 | compute
 8 | ready     | ceph-2              |       6 | 10.20.0.12 | ec:f4:bb:ea:29:c4 | ceph-osd
 9 | ready     | ceph-3              |       6 | 10.20.0.10 | ec:f4:bb:ea:2e:4c | ceph-osd
```

5.6.7 Environment Settings

The illustration below, taken from the environment post-deployment, summarizes environment settings.

Figure 28. Environment settings summary

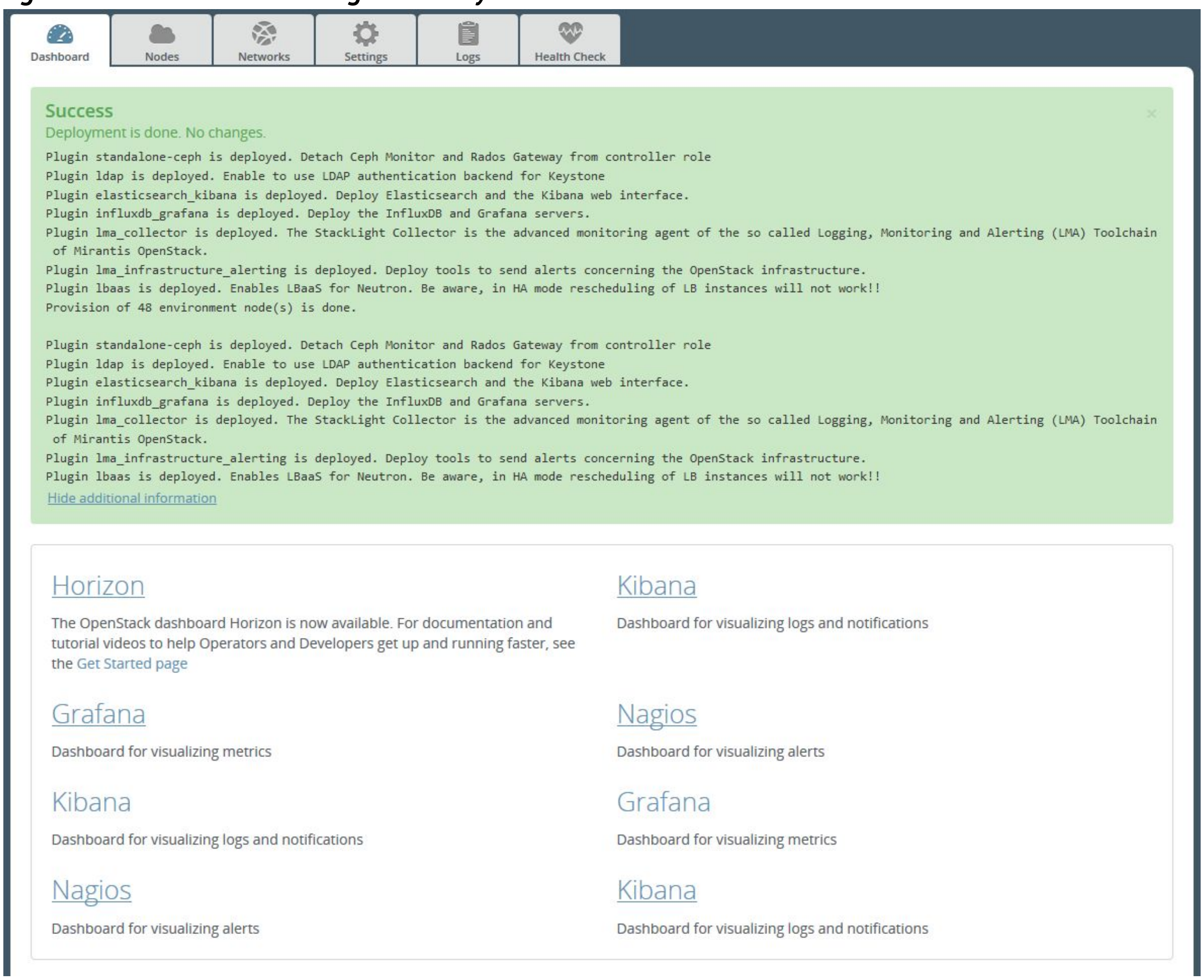
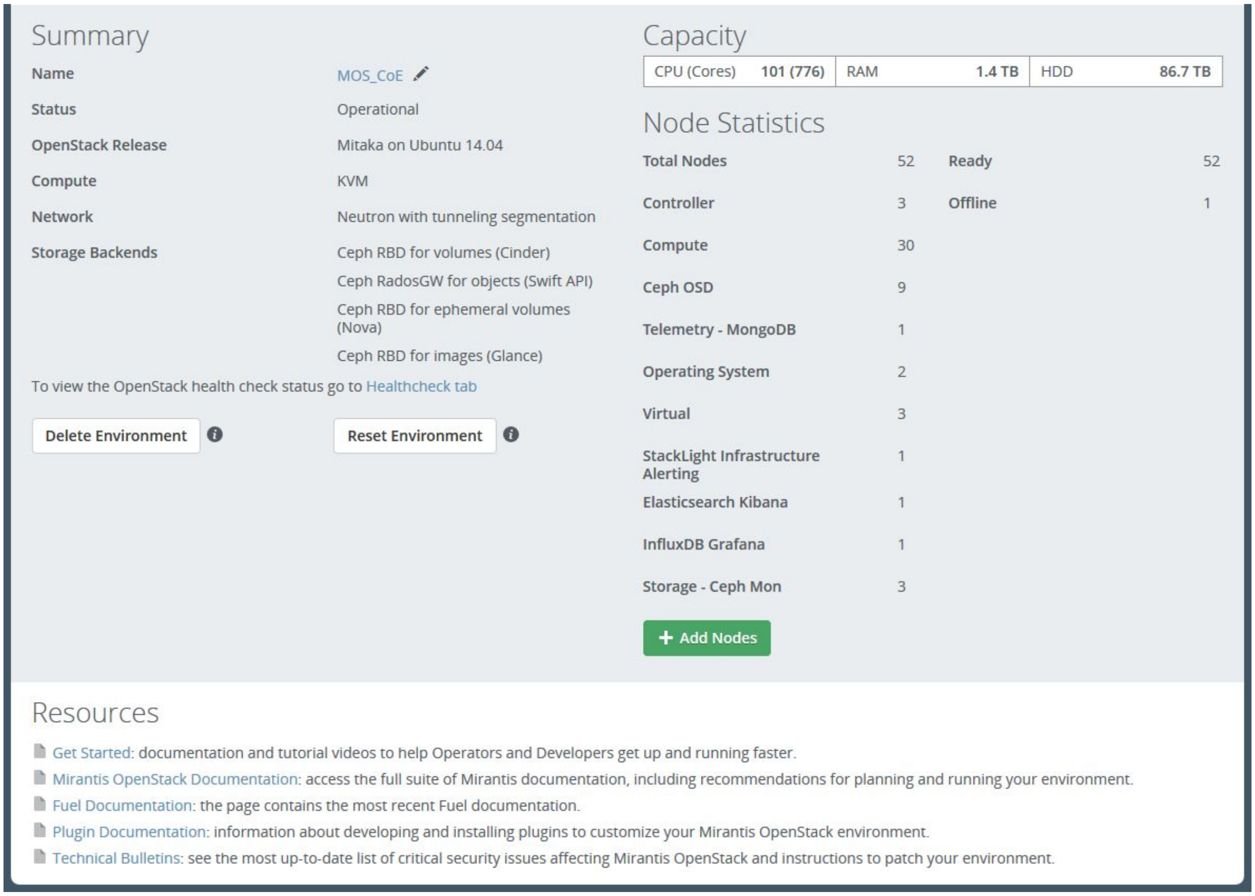


Figure 29. Environment settings summary (continued 1)



5.6.8 Additional Settings

Nodes Network Allocation

Networks for the PoC deployment are defined using Fuel network templates. A sample network template file for the deployment is linked in the Appendices.

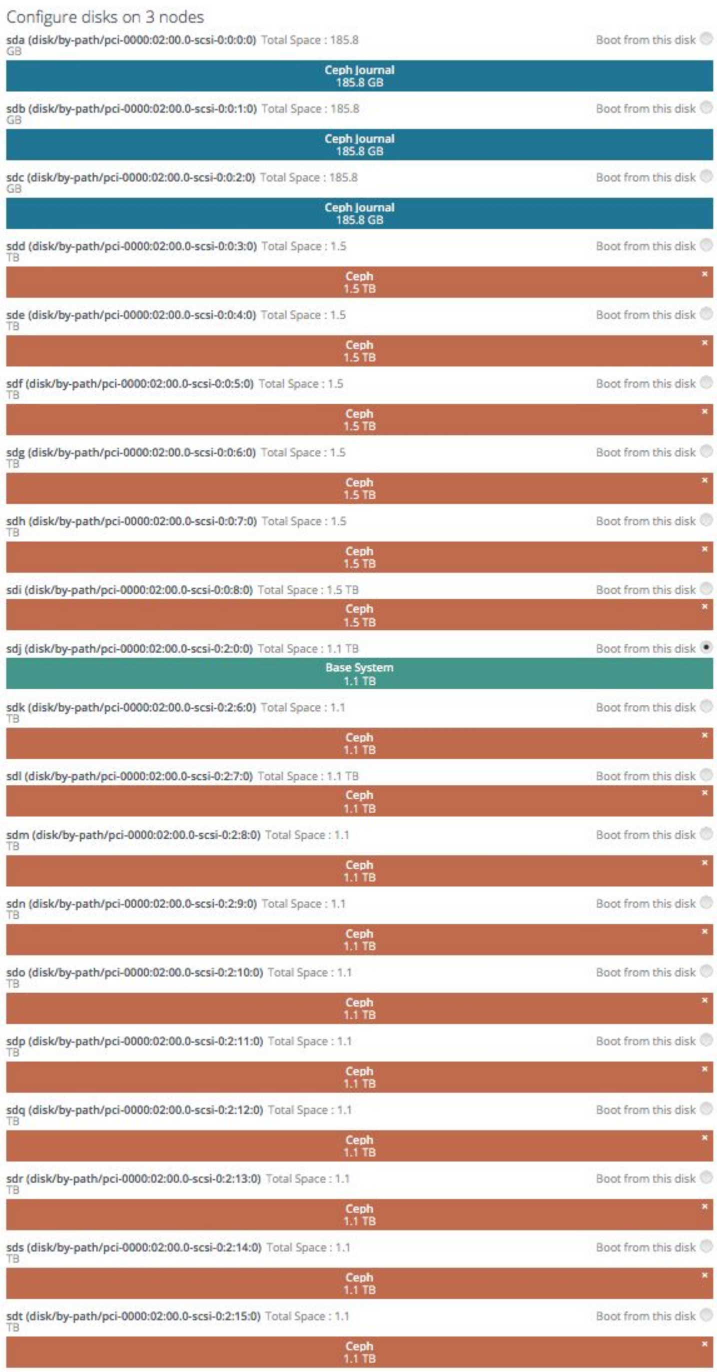
HDD Allocations for Controller, Compute and Storage Nodes

Fuel generally suggests reasonable starting minimums for storage allocation on these nodes. In some cases, it may be necessary to increase the space on the /var/log partition to retain more log data.

Ceph Nodes HDD Allocation

Allocate drive space on Ceph nodes as shown below.

Figure 30. Allocating drive space on Ceph nodes



5.6.9 Common Settings

Set passwords and other required values (our choices are shown) on the sub-panels of Fuel's Settings tab.

Figure 31. Common settings

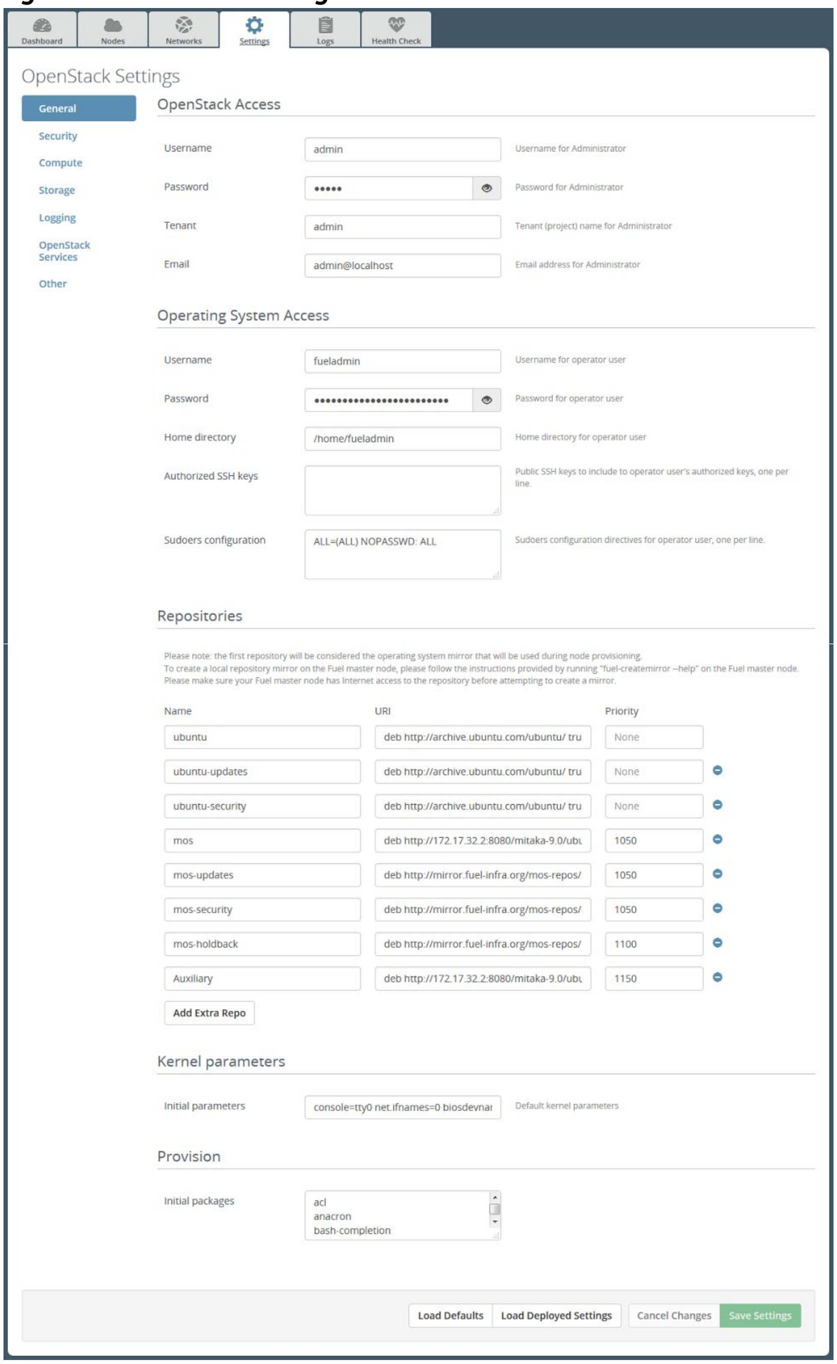


Figure 32. Common settings (continued 1)

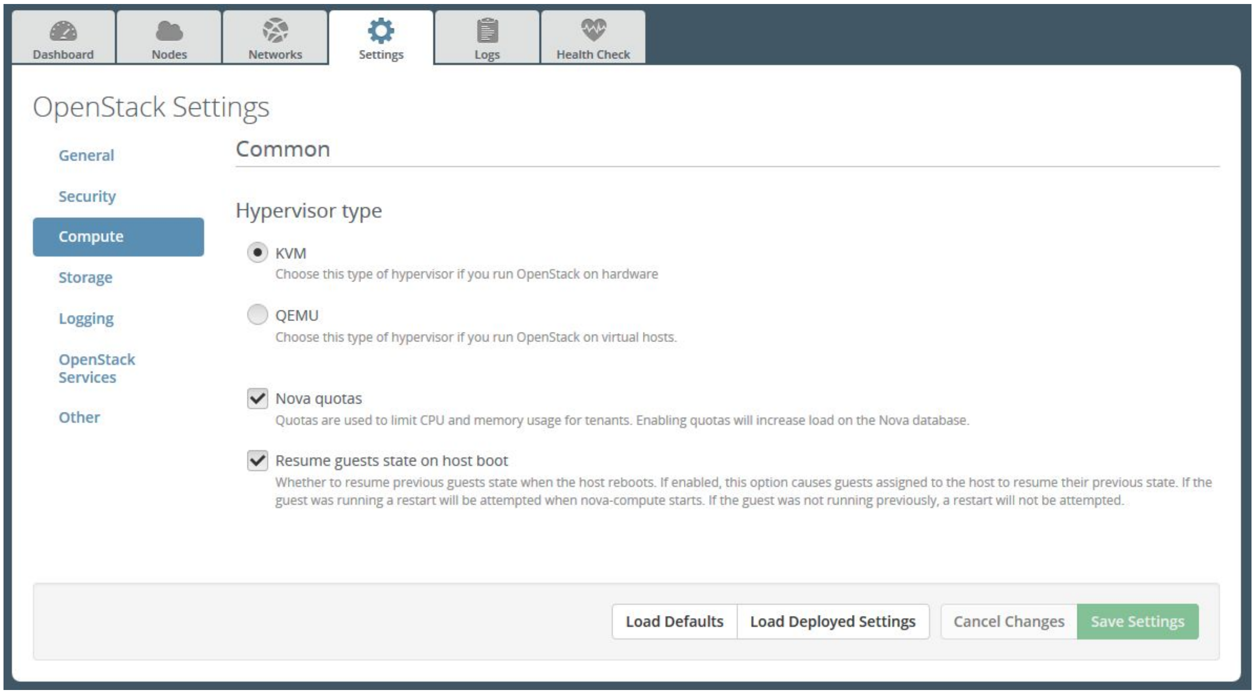


Figure 33. Common settings (continued 2)

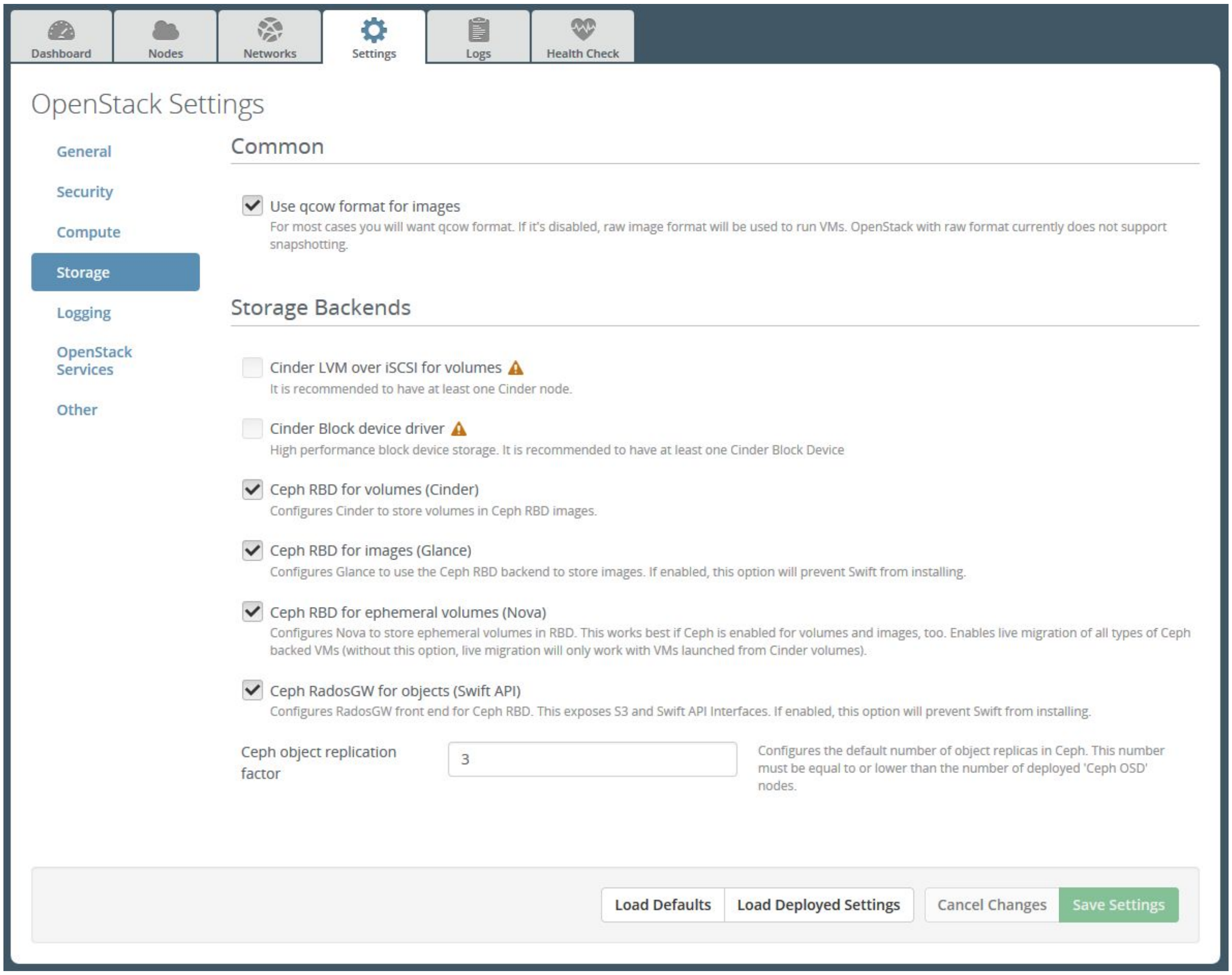


Figure 34. Common settings (continued 3)

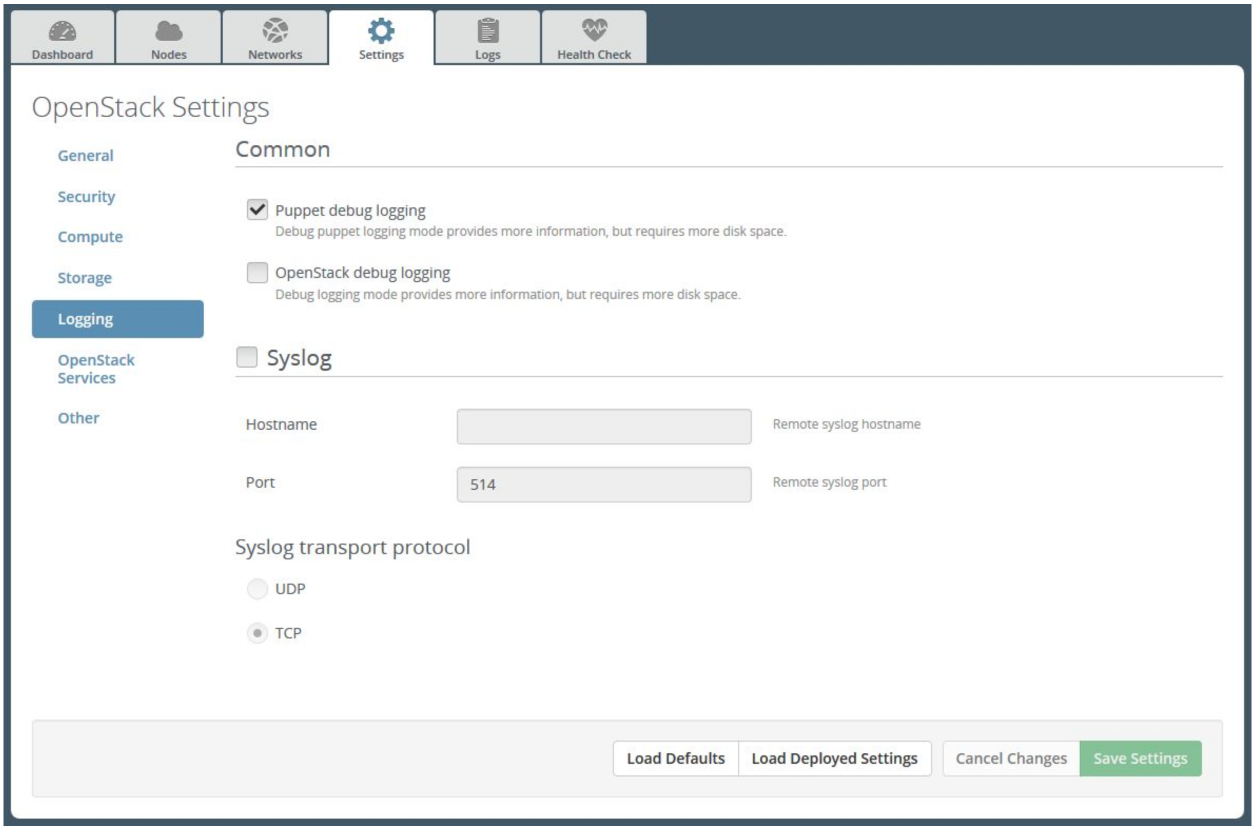
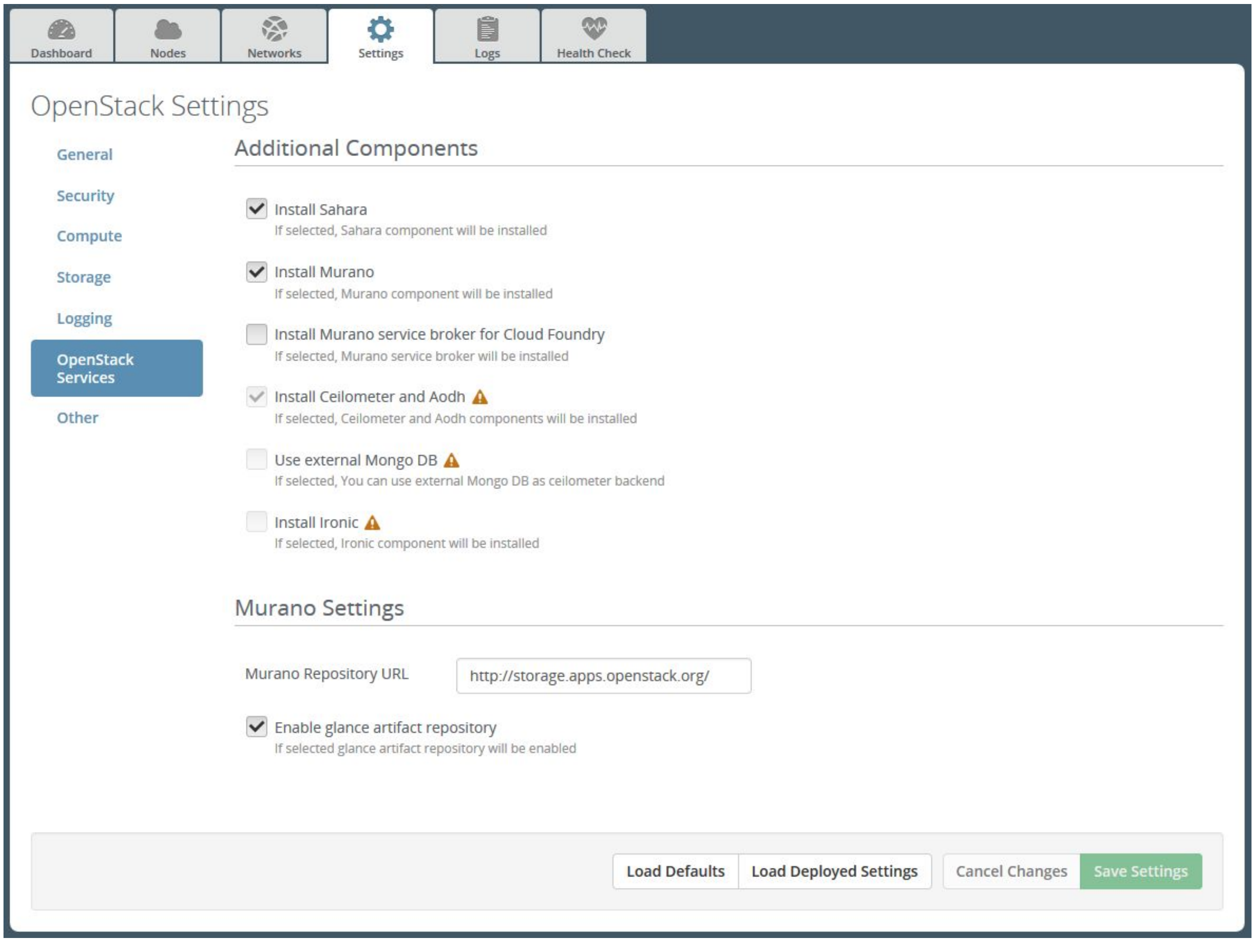
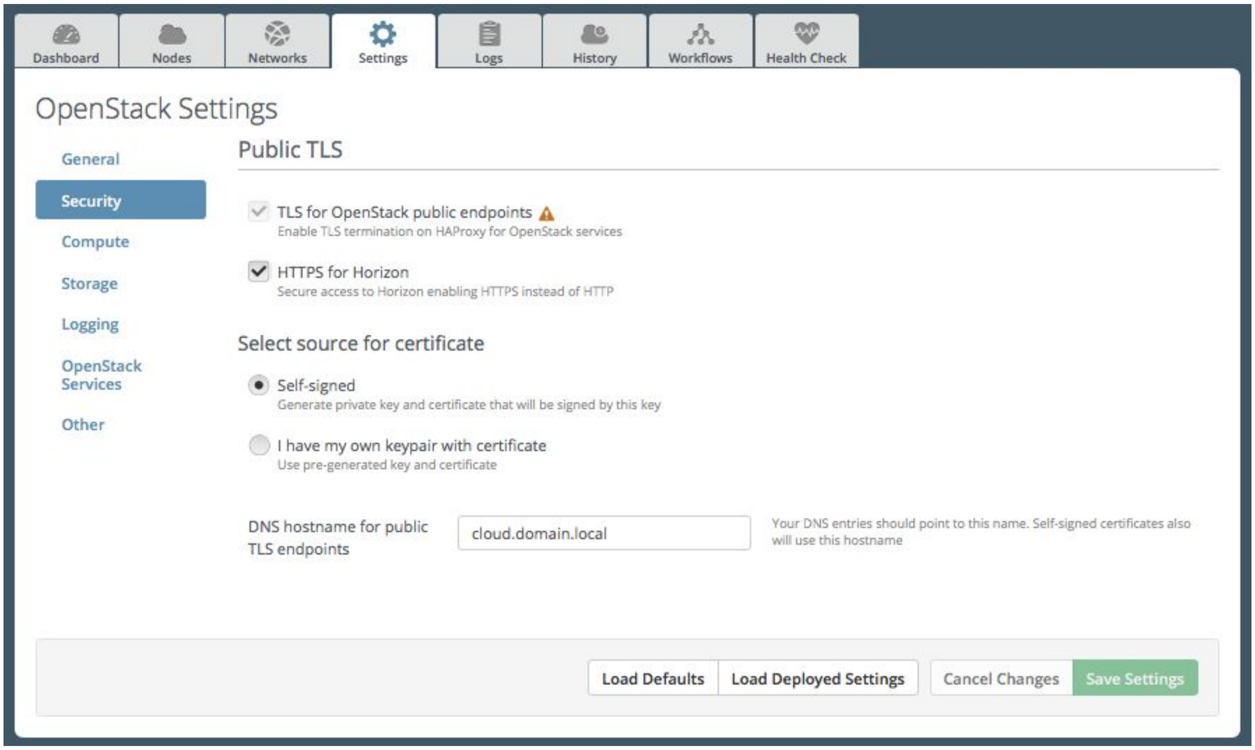


Figure 35. Common settings (continued 4)



5.6.10 TLS Settings

Figure 36. TLS settings



5.6.11 LDAP Plugin Settings

For the sake of simplicity, LDAP was not configured on this environment.

5.6.12 Standalone Ceph Plugin Settings

Figure 37. Ceph plugin settings

5.6.13 StackLight Plugin Settings

Figure 38. StackLight plugin settings

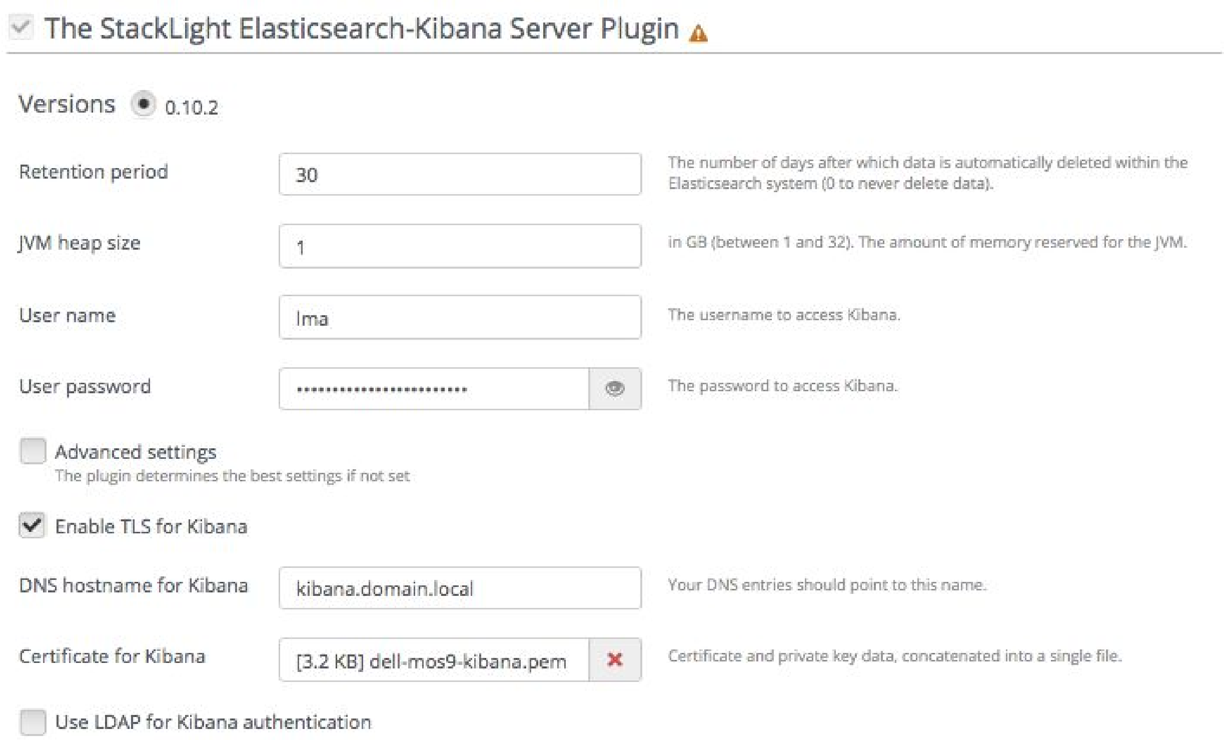


Figure 39. StackLight plugin settings (continued 1)

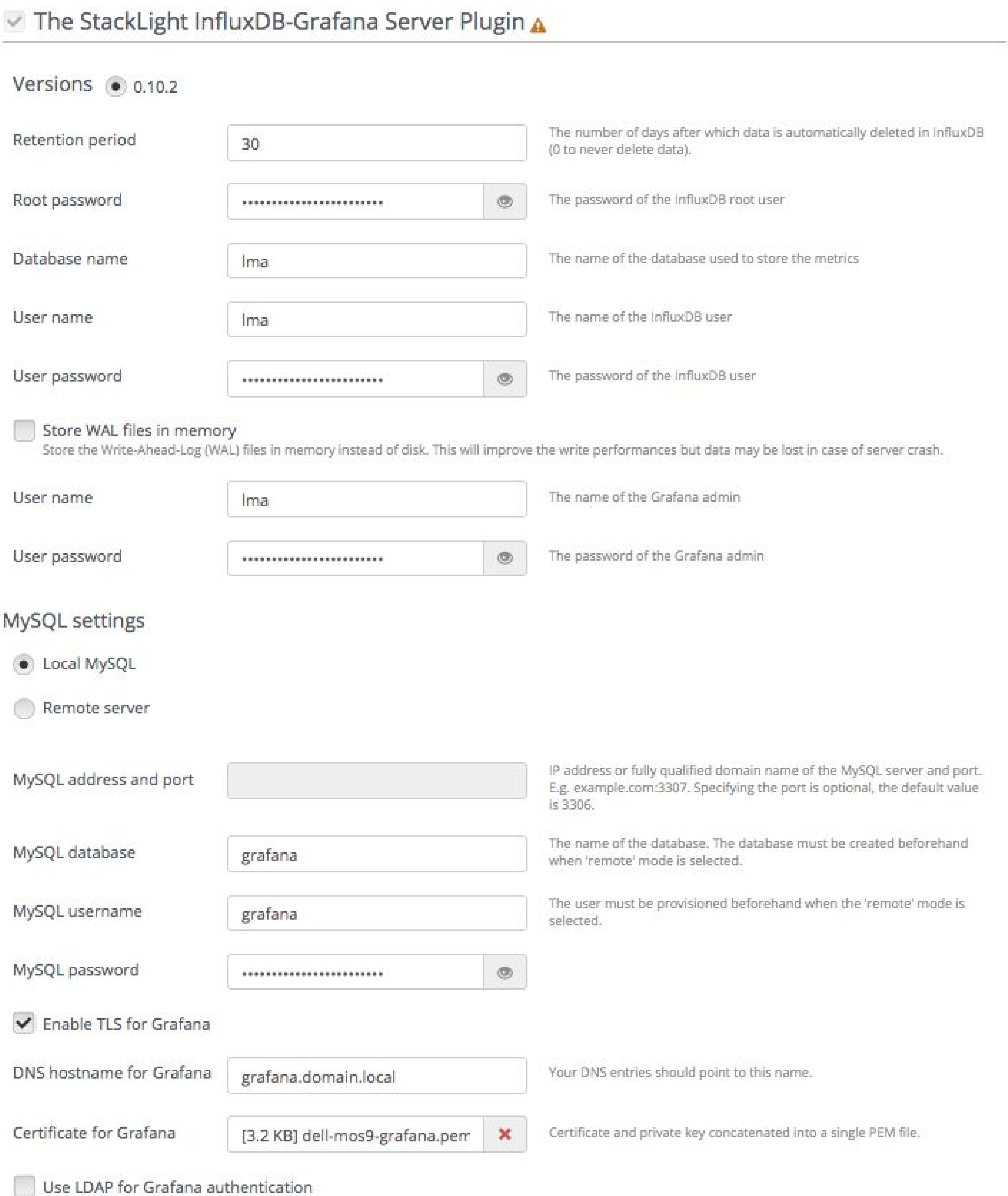


Figure 40. StackLight settings (continued 2)

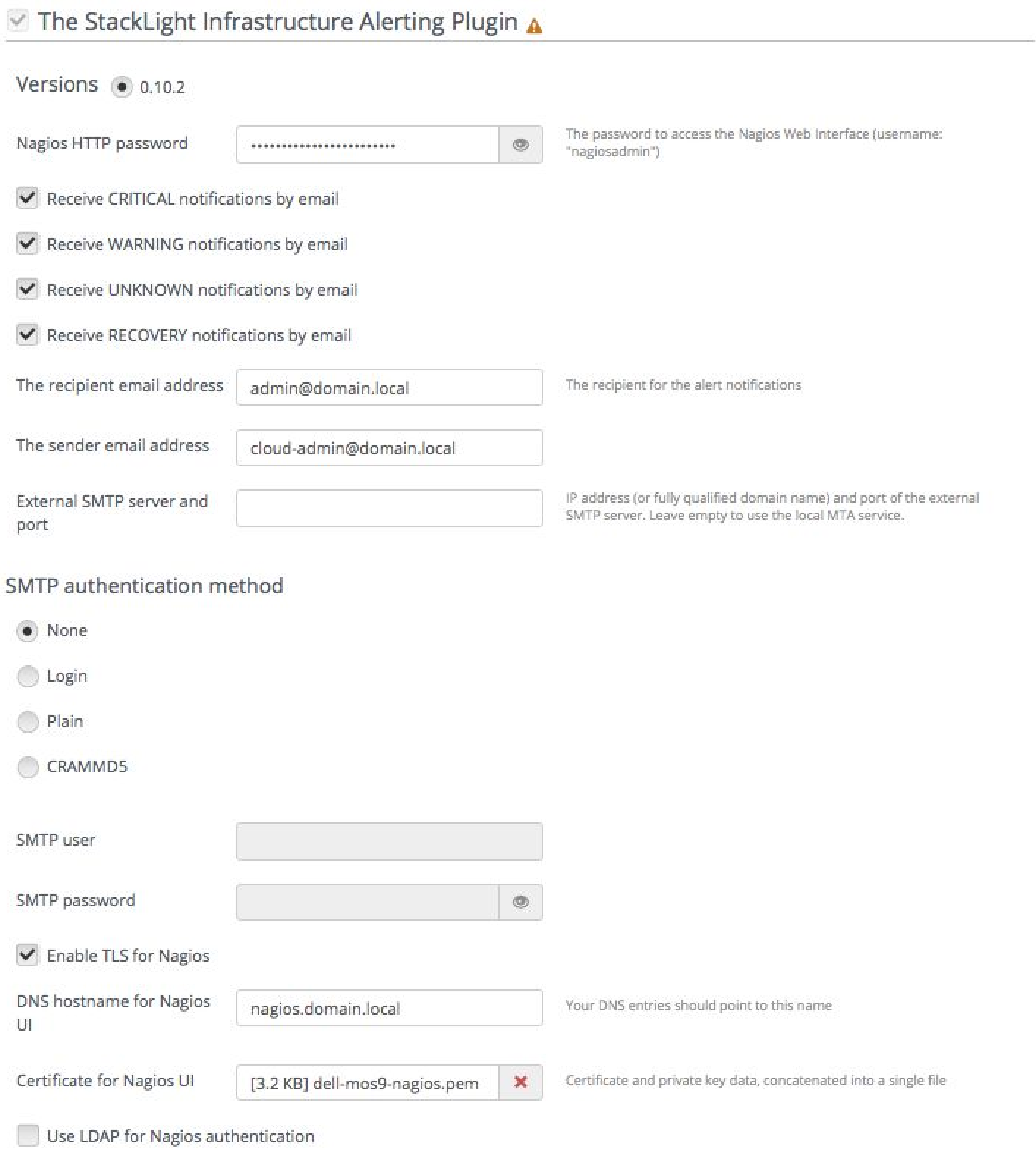
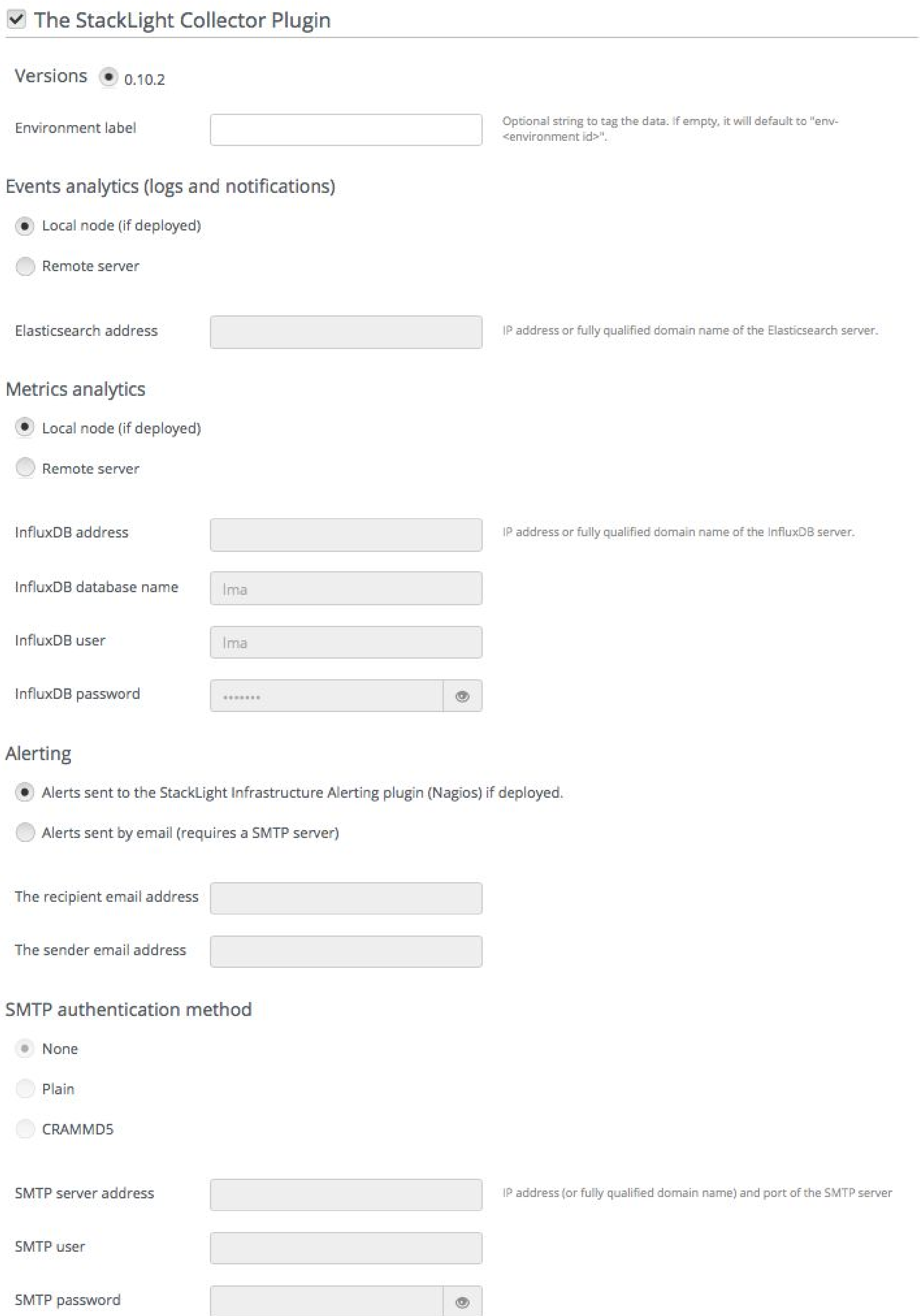


Figure 41. StackLight settings (continued 3)



5.6.14 Deploy the Environment

Deploy the environment using the Fuel web UI or CLI.

```
<fuel_master># fuel --env <ENV_ID> --deploy
```

6 Post-Deployment Customizations

6.1 Recreate OSDs on SSDs

Fuel doesn't differentiate SSDs and SAS drives when placing Ceph OSDs on them. In order to fully utilize SSD bandwidth, however, SSDs they should be split across approximately 5 OSDs. To enable this, you need to remove OSDs created by Fuel on those SSDs, split each SSD into five partitions, and create OSDs on them.

1. Find all OSD numbers you want to replace. Display all OSDs grouped by the hosts they're on by using the "ceph osd tree" command on any Controller or Ceph node. The nodes will have weights close to their size in TB. In the case shown, the weight is 1.45 (for a 1.5TB OSD):

```
root@node-12:~# ceph osd tree
ID  WEIGHT   TYPE NAME          UP/DOWN REWEIGHT PRIMARY-AFFINITY
 -6 34.79999 root root
 -7  8.70000     host node-7
  0  1.45000         osd.0           up  1.00000          1.00000
  1  1.09000         osd.1           up  1.00000          1.00000
  2  1.09000         osd.2           up  1.00000          1.00000
  3  1.09000         osd.3           up  1.00000          1.00000
  4  1.09000         osd.4           up  1.00000          1.00000
  5  1.45000         osd.5           up  1.00000          1.00000
  6  1.09000         osd.6           up  1.00000          1.00000
  7  1.09000         osd.7           up  1.00000          1.00000
  8  1.45000         osd.8           up  1.00000          1.00000
...
```

2. Write down host-to-osd_ids mapping. Here, we store them as shell variables:

```
host7_osd='0 5 8 15 19 20'
host8_osd='21 22 27 29 31 39'
host9_osd='26 41 44 48 54 64'
host10_osd='33 63 70 72 77 79'
```

3. SSH to the first Ceph OSD node. Stop and remove the OSDs on it:

```
osd="$host7_osd"
for i in $osd; do
  service ceph-osd stop id=$i
  sleep 1
  ceph osd crush remove osd.$i
  ceph auth del osd.$i
  ceph osd rm $i
done
```

```
disks=$(mount|egrep "ceph-(${osd// /|}) "|cut -d' ' -f1|cut -d/ -f3|tr '\n' ' '|tr -d
3)
for i in $osd; do umount /var/lib/ceph/osd/ceph-$i; done
```

Note that you already have a shell variable called "disks," containing disk names of SSDs used by these OSDs. We'll need this variable for the next steps.

4. Repartition all SSDs listed in the "disks" variable. Create five equal XFS partitions on each SSD.

```
for i in $disks; do
parted /dev/$i rm 3
parted /dev/$i mkpart ceph xfs 237MB 320253MB
parted /dev/$i mkpart ceph xfs 320253MB 640269MB
parted /dev/$i mkpart ceph xfs 640269MB 960285MB
parted /dev/$i mkpart ceph xfs 960285MB 1280301MB
parted /dev/$i mkpart ceph xfs 1280301MB 1600321MB
done
```

5. Create and activate new OSDs on the new partitions.

```
for d in $disks; do for i in 3 4 5 6 7; do ceph-disk -v prepare --fs-type xfs
--cluster ceph -- $d$i; done; done
for d in $disks; do for i in 3 4 5 6 7; do ceph-disk -v activate --mark-init upstart
--mount $d$i; done; done
```

6. Check Ceph status using the "ceph status" command. Wait until the Ceph cluster is healthy again and all OSDs are in the "active+clean" state.

```
root@node-12:~# ceph status
    cluster ee126587-f638-484f-a23a-7aac7e931c12
     health HEALTH_OK
     monmap e3: 3 mons at
{node-14=192.168.3.14:6789/0,node-15=192.168.3.15:6789/0,node-22=192.168.3.17:6789/0}
            election epoch 14, quorum 0,1,2 node-14,node-15,node-22
     osdmap e4426: 180 osds: 178 up, 178 in
      pgmap v738789: 8960 pgs, 13 pools, 36966 MB data, 9582 objects
            373 GB used, 100929 GB / 101302 GB avail
                8960 active+clean
```

7. Repeat steps 3-6 for for each Ceph OSD node.

6.2 Distribute SSDs and SAS Drives to Different Pools

We're going to put the RadosGW pool on our SAS drives while keeping the other pools (for Cinder, Nova, Glance) on our SSDs. In order to do this we need to move the SSD OSDs into different "roots" (in terms of the Ceph CRUSH Map), create two rulesets to distinguish these two "roots", and assign the rulesets to pools.

- Obtain the CRUSH Map, following instructions in the official Ceph manual. A sample CRUSH Map is shown in the Appendices to this document.
- Create a second root and hosts attached to it. Put SSD OSDs into those hosts.
- Create a second rule and adjust both rules in the way that each of them points to a different root (adjust the "step take" stanza).

- Make sure that the old rule retains its ID (usually "0") and points to a root with SAS drives while the new rule points to a root with SSDs.
- Compile and upload the modified CRUSH Map.
- Change ruleset ID for Glance, Cinder, and Nova pools:

```
for pool in compute images volumes: do
ceph osd pool set $pool crush_ruleset 4
done
```

- Verify that Nova, Glance, and Cinder still work.

6.3 Migrate Fuel Master Node to the Deployed Cloud

Use the fuel-migrate script to migrate the Fuel Master node to a virtual machine on a compute node. This allows for reduced resource utilization in small environments and lets the Fuel Master node run on physical or virtual machines by making it host agnostic.

To run the script, issue the following command:

```
<fuel_master># fuel-migrate
```

More info about migrating Fuel node to the cloud can be found here.

6.4 Adding and Removing Compute and Storage Nodes

Mirantis (via openstack.org) maintains a guide for adding or removing compute and storage nodes from an environment.

IMPORTANT: Before removing a compute node, all VMs running on that node need to be migrated to other compute nodes.

Testing the Deployed Cloud

A test plan and complete test report for the *Compact Cloud* are available for online review.

Resources

Mirantis OpenStack 9.1 Documentation
Mirantis OpenStack Planning Guide
Fuel Installation Guide
Fuel User Guide
Dell Power Consumption Calculators

Appendix A

Network Template Example

A sample network template for Compact Cloud is available for review online.

Ceph CRUSH Map Example

A sample Ceph CRUSH map is available for review online.

Component Configuration Details

OpenLDAP-backed Keystone

Keystone Identity can use OpenLDAP as a cache proxy for authentication requests to existing external Directory servers (MS AD, OpenLDAP, etc), improving responsiveness and decreasing load on Directory servers. In this case, each controller runs an OpenLDAP cache (proxy) server connected to an external AD/LDAP server. HAProxy controls the OpenLDAP endpoint on a VIP.

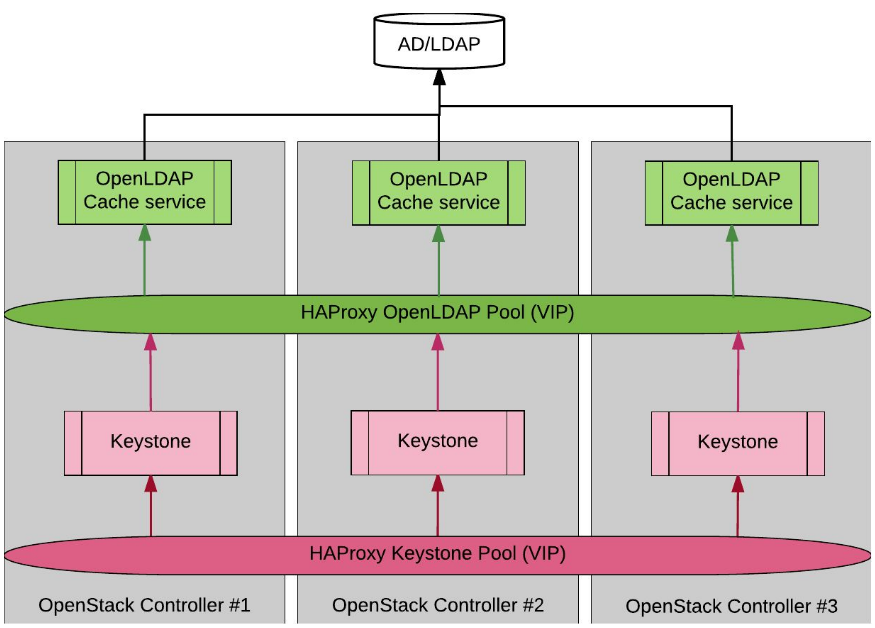


Figure 42. Compact Cloud - OpenLDAP backed Keystone, HA diagram

Using Ceph Block Devices with Nova

To use Ceph block devices by default with Nova, configure Glance according to the Ceph documentation: http://docs.ceph.com/docs/master/rbd/rbd-openstack/#configuring-nova

Using Ceph Block Devices with Glance

To use Ceph block devices by default with Glance, configure Glance according to the Ceph documentation: http://docs.ceph.com/docs/master/rbd/rbd-openstack/#configuring-glance

Using Ceph Block Devices with Cinder

To use Ceph block devices by default with Cinder, and to enable the cinder-backup feature, configure Cinder according to the Ceph documentation:
http://docs.ceph.com/docs/master/rbd/rbd-openstack/#configuring-cinder,
http://docs.ceph.com/docs/master/rbd/rbd-openstack/#configuring-cinder-backup

Ceilometer Meters and Events

Below are enumerated OpenStack Ceilometer meters and events we recommend collecting using the StackLight subsystem specified as a component of *Compact Cloud*.

Recommended Ceilometer Meters

- cpu
- cpu.util
- disk.read.bytes
- disk.write.bytes
- image
- image.download
- image.upload
- memory
- network.incoming.bytes
- network.outgoing.bytes
- vcpus
- volume
- Volume.size

Recommended Ceilometer Events

- compute.instance.create.end
- compute.instance.create.start
- compute.instance.delete.end
- compute.instance.delete.start
- compute.instance.pause.end

- compute.instance.pause.start
- compute.instance.power_off.end
- compute.instance.power_off.start
- compute.instance.power_on.end
- compute.instance.power_on.start
- compute.instance.rebuild.end
- compute.instance.rebuild.start
- compute.instance.resize.confirm.end
- compute.instance.resize.confirm.start
- compute.instance.resume
- compute.instance.suspend
- compute.instance.suspstart
- compute.instance.unpause.end
- compute.instance.unpause.start
- compute.instance.update.end
- compute.instance.update.start
- compute.reboot.end
- compute.reboot.start
- floatingip.create.end
- floatingip.create.start
- image.delete
- image.update
- image.upload
- network.create.end
- network.create.start
- network.services.firewall.create.end
- network.services.firewall.create.start
- network.services.firewall.delete.end
- network.services.firewall.delete.start
- network.services.lb.vip.create.end
- network.services.lb.vip.create.start
- network.services.lb.vip.delete.end
- network.services.lb.vip.delete.start
- network.update.end
- network.update.start
- port.create.end
- port.create.start
- port.delete.end
- port.delete.start
- router.create.end
- router.create.start
- router.update.end
- router.update.start
- snapshot.create.end

- snapshot.create.start
- snapshot.delete.end
- snapshot.delete.start
- subnet.create.end
- subnet.create.start
- subnet.delete.end
- subnet.delete.start
- volume.create.end
- volume.create.start
- volume.delete.end
- volume.delete.start
- volume.resize.end
- volume.resize.start
- volume.update.end
- volume.update.start

Reference Information - OpenStack

OpenStack is a cloud operating system that controls large pools of compute, storage, and networking resources throughout a datacenter, all managed through a dashboard that gives administrators control while empowering their users to provision resources through a web interface.

OpenStack Components

Table 13. OpenStack core projects

Name	Purpose	Description
Nova[2]	Compute service	Manages the lifecycle of compute instances in an OpenStack environment. Responsibilities include spawning, scheduling and decommissioning of machines on demand.
Neutron[3]	Networking service	Enables network connectivity as a service for other OpenStack services, such as OpenStack Compute. Provides an API for users to define networks and the attachments into them. Has a pluggable architecture that supports many popular networking vendors and technologies.
Swift[4]	Object Storage	Stores and retrieves arbitrary unstructured data objects via a RESTful, HTTP based API. It is highly fault tolerant with its data replication and scale out architecture. Its implementation is not like a file server with mountable directories.
Cinder[5]	Block Storage	Provides persistent block storage to running instances. Its pluggable driver architecture facilitates the creation and management of block storage devices.
Keystone[6]	Identity service	Provides an authentication and authorization service for other OpenStack services. Provides a catalog of endpoints for all OpenStack services.

[2] Nova WIKI - https://wiki.openstack.org/wiki/Nova
[3] Neutron WIKI - https://wiki.openstack.org/wiki/Neutron
[4] Swift WIKI - https://wiki.openstack.org/wiki/Swift
[5] Cinder WIKI - https://wiki.openstack.org/wiki/Cinder
[6] Keystone WIKI - https://wiki.openstack.org/wiki/Keystone

Table 13. OpenStack core projects - Continued		
Glance[7]	Image service	Stores and retrieves virtual machine disk images. OpenStack Compute makes use of this during instance provisioning.

Table 14. OpenStack optional services

Name	Purpose	Description
Horizon[8]	Dashboard	Provides a web based user interface to OpenStack services including Nova, Swift, Keystone, etc.
Ceilometer[9]	Telemetry	Collects data on the utilization of the physical and virtual resources comprising deployed clouds, persists these data for subsequent retrieval and analysis, and triggers actions when defined criteria are met.
Heat[10]	Orchestration	Provides a human- and machine-accessible service for managing the entire lifecycle of infrastructure and applications within OpenStack clouds.
Trove[11]	Database	Provides scalable and reliable Cloud Database as a Service provisioning functionality for both relational and non-relational database engines
Sahara[12]	Elastic Map Reduce	Provides a simple means to provision a data-intensive application cluster (Hadoop or Spark) on top of OpenStack
Ironic[13]	Bare-Metal Provisioning	Provides bare metal machines instead of virtual machines.
Zaqar[14]	Messaging	Provides a multi-tenant cloud messaging service for web and mobile developers.

[7] Glance WIKI - https://wiki.openstack.org/wiki/Glance
[8] Horizon WIKI - https://wiki.openstack.org/wiki/Horizon
[9] Ceilometer WIKI - https://wiki.openstack.org/wiki/Telemetry
[10] Heat WIKI - https://wiki.openstack.org/wiki/Heat
[11] Trove WIKI - https://wiki.openstack.org/wiki/Trove
[12] Sahara WIKI - https://wiki.openstack.org/wiki/Sahara
[13] Ironic WIKI - https://wiki.openstack.org/wiki/Ironic
[14] Zaqar WIKI - https://wiki.openstack.org/wiki/Zaqar

Table 14. OpenStack optional services - Continued		
Monasca[15]	Monitoring	Provides monitoring-as-a-service solution integrated to OpenStack.
Manila[16]	Shared Filesystems	Provides shared file system -as-a-service solution integrated to OpenStack.
Designate[17]	DNS	Provides DNS -as-a-service solution integrated to OpenStack.
Barbican[18]	Key Management	Provides the secure storage, provisioning and management of secrets such as passwords, encryption keys and X.509 Certificates.
Magnum[19]	Containers	Provides the container orchestration engines such as Docker and Kubernetes available as first class resources in OpenStack.
Murano[20]	Application Catalog	Provides an application catalog to OpenStack, enabling application developers and cloud administrators to publish various cloud-ready applications in a browsable categorized catalog.
Congress[21]	Governance	Provides a policy as a service across any collection of cloud services in order to offer governance and compliance for dynamic infrastructures.
Rally[22]	Benchmarking	Provides the toolchain for cloud verification, benchmarking, and profiling.
Mistral[23]	Workflow	Provides business processes workflow -as-a-service solution integrated to OpenStack.

[15] Monasca WIKI - https://wiki.openstack.org/wiki/Monasca
[16] Manila WIKI - https://wiki.openstack.org/wiki/Manila
[17] Designate WIKI - https://wiki.openstack.org/wiki/Designate
[18] Barbican WIKI - https://wiki.openstack.org/wiki/Barbican
[19] Magnum WIKI - https://wiki.openstack.org/wiki/Magnum
[20] Murano WIKI - https://wiki.openstack.org/wiki/Murano
[21] Congress WIKI - https://wiki.openstack.org/wiki/Congress
[22] Rally WIKI - https://wiki.openstack.org/wiki/Rally
[23] Mistral WIKI - https://wiki.openstack.org/wiki/Mistral

OpenStack API Versions

Mirantis OpenStack supports the following versions of the OpenStack API.

Table 15. OpenStack API versions supported in Mirantis OpenStack

Component	API Version
Keystone	v2
Nova	v2
Glance	v2
Cinder	v2
Swift (via Ceph RadosGW)	v1
Neutron	v2
Ceilometer	v2
Murano	v1
Heat	v1

Appendix B - Mirantis Software

Mirantis OpenStack is 100% open source software. Mirantis OpenStack release 9.1 (corresponding to OpenStack Mitaka) is an update to Mirantis OpenStack 9.0, applied by updating a Fuel 9.0 master node as detailed above in 5.4 Fuel Master Node Installation.

Mirantis currently offers several tiers of enterprise-grade, per-node support subscriptions for Mirantis OpenStack 9.x, and will support Mirantis OpenStack 9.x for up to three years for 24 x 7 support tier customers.

Mirantis Managed Offering (MMO) provides design, deployment, scaling, management, monitoring, regular upgrades, and comprehensive support for Mirantis open cloud software at enterprise premise(s) or colocation facility(-ies) of choice, letting customers focus on accelerating their business instead of operating their cloud, and helping them benefit from continuous open source innovation without disruption or risk. Base pricing for MMO is shown below.

The following table of Dell EMC SKUs:

- Applies to US only
- Category is 'Licensing Support'
- Segment is ENT (Enterprise)
- Manufacturer is Mirantis
- Subclass Code is 308
- Currency Type is USD

SKU	Part Num.	Description
A9226527	MOS-001-24X7	Mirantis - OpenStack Subscriptions per Node; 1 -Year 24x7 Support
A9226528	MOS-001-8X5	Mirantis - OpenStack Subscriptions per Node; 1 -Year 8x5 Support
A9226911	MMO-001-BASE	Mirantis Managed OpenStack Services