Contents

Background

There are two common ways of installing OpenStack, manually or by using automation tools. There is much focus on the full automation of OpenStack deployments using tools such as Puppet, Chef, JuJu and others. While these tools offer great advantages over manual configuration, they do hide the OpenStack installation and configuration details. This document can be used by those interested in learning more about the OpenStack Grizzly High-Availability (HA) installation process or for those not interested in using automation tools to deploy HA. The document covers the following OpenStack software components:

- Glance (Image Service)
- Keystone (Identity Service)
- Nova (Compute Service)
- Horizon (OpenStack Dashboard Web User Interface)
- Quantum (Network Service)
- Cinder (Block Storage Service)
- Swift (Object Storage Service)

High-Availability Introduction

Most OpenStack deployments are maturing from evaluation-level environments to highly available and highly scalable environments to support production applications and services. The Cisco OpenStack High-Availability Guide differs from the OpenStack High Availability Guide by providing an active/active, highly scalable model for OpenStack deployments. The architecture consists of the following components used to provide high-availability to OpenStack services:

- MySQL Galera is synchronous multi-master cluster technology for MySQL/InnoDB databases that includes features such as:
    - Synchronous replication
    - Active/active multi-master topology
    - Read and write to any cluster node
    - True parallel replication, on row level
    - Direct client connections, native MySQL look & feel
    - No slave lag or integrity issues

- Several OpenStack services utilize a message queuing system to send and receive messages between software components. The Cisco reference architecture leverages RabbitMQ as the messaging system since it is most commonly used within the OpenStack community. RabbitMQ Clustering and RabbitMQ Mirrored Queues provide active/active and highly scalable message queuing for OpenStack services.
    - RabbitMQ Clustering: If your RabbitMQ broker consists of a single node, then a failure of that node will cause downtime, temporary unavailability of service, and potentially loss of messages. A cluster of RabbitMQ nodes can be used to construct your RabbitMQ broker. Clustering RabbitMQ nodes are resilient to the loss of individual nodes in terms of the overall availability of service. All data/state required for the operation of a RabbitMQ broker is replicated across all nodes, for reliability and scaling. An exception to this are message queues, which by default reside on the node that created them, though they are visible and reachable from all nodes.
    - RabbitMQ Mirrored Queues: While exchanges and bindings survive the loss of individual nodes, message queues and their messages do not. This is because a queue and its contents reside on exactly one node, thus the loss of a node will render its queues unavailable. To solve these various problems, RabbitMQ has developed active/active high-availability for message queues. This works by allowing queues to be mirrored on other nodes within a RabbitMQ cluster. The result is that should one node of a cluster fail, the queue can automatically switch to one of the mirrors and continue to operate, with no unavailability of service. This solution still requires a RabbitMQ Cluster, which means that it will not cope seamlessly with network partitions within the cluster and, for that reason, is not recommended for use across a WAN (though of course, clients can still connect from as near and as far as needed).
- HAProxy and Keepalived provide load-balancing betwwen clients and OpenStack API Endpoints.
    - HAProxy is a free, very fast and reliable software solution offering high availability, load balancing, and proxying for TCP and HTTP-based applications. HAProxy implements an event-driven, single-process model which enables support for a high number of simultaneous

connections.

- Keepalived is a routing software written in C. The main goal of the Keepalived project is to provide simple and robust facilities for load-balancing and high-availability to Linux systems and Linux-based infrastructures. Load-balancing frameworks rely on the well-known and widely used Linux Virtual Server (IPVS) kernel module providing Layer4 load-balancing. Keepalived implements a set of checkers to dynamically and adaptively maintain and manage load-balanced server pool according their health. On the other hand high-availability is achieved by VRRP protocol. VRRP is a fundamental brick for router fail-over.

- Multiple Quantum L3 and DHCP Agents Blueprint, as it states, allows multiple Quantum Layer-3 and DHCP Agents to be deployed for high-availability and scalability purposes. At this time (Grizzly release), multiple DHCP Agents can service a Quantum network, however, only a single L3 Agent can service one Quantum network at a time. Therefore, the L3 Agent is a single point of failure and is not included in the Cisco High-Availability Deployment Guide. Quantum Provider Network Extensions are used to map physical data center networks to Quantum networks. In this deployment model, Quantum relies on the physical data center to provide Layer-3 high-availability instead of the L3 Agent.
- Glance uses Swift as the back-end to storage OpenStack images. Just as with the rest of the OpenStack API's, HAProxy and Keepalived provide high-availability to the Glance API and Registry endpoints.
- Swift: Multiple Swift Proxy nodes are used to provide high-availability to the Swift proxy service. Replication provides high-availability to data stored within a Swift object-storage system. The replication processes compare local data with each remote copy to ensure they all contain the latest version. Object replication uses a hash list to quickly compare subsections of each partition, and container and account replication use a combination of hashes and shared high water marks.

Dependencies

Critical Reminders

The most common OpenStack HA deployment issues are either incorrect configuration files or not deploying the nodes in the proper order. To save you from future troubleshooting steps, ENSURE that you deploy the nodes in the order described within the document and verify the accuracy of all configuration files. You will likely be using your own IP addressing and passwords in your setup and it is critical to ensure any variations from this guide are fully understood.

Do not configure RAID on the hard disks of Swift Storage Nodes. Swift performs better without RAID and disk redundancy is unneeded since Swift protects the data through replication. Therefore, if a RAID Controller manages the hard disks, ensure you present each of the hard disks independently. Our example uses disk /dev/sda for the Operating System installation and disks /dev/sdb-/dev/sdf for Swift storage. Please remember to modify these definitions based on your specific deployment environment. Additional Swift considerations and tuning information can be found here.

Compute Nodes run Cinder Volume to provide block storage services to Instances. The default Cinder driver (volume_driver=nova.volume.driver.ISCSIDriver) is an iSCSI solution that employs the use of Linux Logical Volume Manager (LVM). Therefore, you must create an LVM Volume Group either during the Ubuntu Precise installation or afterwards. The name of the LVM Volume Group must match the volume_group definition in cinder.conf. Our example uses the name nova-volumes for the LVM Volume Group and associated cinder.conf volume_group name.

The password used in our examples is keystone_admin. Every account, service and configuration file uses this one password. You will want to change this in your setup and you certainly want to use a strong password and a different password for each account/service if this system is going into production.

Operating System

The operating system used by OpenStack nodes in the HA architecture is Ubuntu 12.04 LTS (Precise). Before installing the operating system, RAID should be configured for hard disks of all nodes, except Swift storage nodes. The RAID type will depend on your deployment needs and RAID Controller specifications. At the minimum, select a RAID type that provides redundancy in the event of a single disk failure. The details for configuring RAID on UCS B-Series and C-Series servers can be found here

Server Requirements

Our deployment uses 13 Cisco UCS C-series servers to serve the roles of Controller, Compute, Load-Balancer and Swift Proxy/Storage. The environment scales linearly, therefore individual nodes can be added to increase capacity for any particular OpenStack service. The five distinct node types used in this document are:

- **3 Controller Nodes-** Runs Nova API, Nova Conductor, Nova Consoleauth, Nova Novncproxy, Nova Scheduler, NoVNC, Quantum Server, Quantum Plugin OVS, Glance API/Registry, Keystone, Cinder API, Cinder Scheduler, OpenStack Dashboard, RabbitMQ Server, MySQL Server WSREP and Galera.
    - ♦ Provides management functionality of the OpenStack environment.

- **3 Compute Nodes-** Runs Nova Compute, Quantum OVS and DHCP Agents, Cinder Volume and TGT services.
    - ♦ Provides the hypervisor role for running Nova instances (Virtual Machines) and presents LVM volumes for Cinder block storage.

- **2 Load-Balancer Nodes-** Runs HAProxy and Keepalived to load-balance traffic across Controller and Swift Proxy clusters.

- **2 Swift Proxy Nodes-** The Proxy Node is responsible for tying together users and their data within the the Swift object storage system. For each request, it will look up the location of the account, container or object in the Swift ring and route the request accordingly. The public API is also exposed by Proxy Node.

- **3 Swift Storage Nodes-** Each Storage Nodes contains Swift object, container, and account services. At a very high-level, these are the servers that contain the user data and perform replication among one another to keep the system in a consistent state.

Networking Requirements

The OpenStack HA environment uses five separate networks. Three of the five networks are used by Tenants. Three tenant networks are being used as an example, and thus the tenant networks can be increased or decreased based on your deployment needs. Connectivity within Tenants uses Quantum with the Open vSwitch (OVS) plugin and Provider Network Extensions. Provider Network Extensions allow cloud administrators to create OpenStack networks that map directly to physical networks in the data center and support local, VLAN and GRE deployment models. Our example uses the Provider VLAN networking model. The network details are as follows:

- **1 Management Network**
    - ♦ This network is used to perform management functions against the node. For example, SSH'ing to the nodes to change a configuration setting. The network is also used for lights-out management using the CIMC interface of the UCS servers. Lastly, OpenStack

API's and the Horizon web dashboard is associated to this network.
- An IP address for each node is required for this network. If using lights-out management such as CIMC, each node will require 2 addresses from this network.
- This network typically employs private (RFC1918).

- **3 Tenant Networks**
  - These networks are used to provide connectivity to Instances. Since Quantum Provider Networking Extensions are being used, it is common to give tenants direct access to a "public" network that can be used to reach the Internet.
  - Compute Nodes will have an interface attached to this network. Since the Compute Node interfaces that attach to this network are managed by OVS, they should not contain an IP address.
  - This network typically employs publicly routable IP addressing if external NAT'ing is not used upstream towards the Internet edge (**Note:** in this document all IP addressing for all interfaces comes out of various private addressing blocks).
- **1 Storage Network**
  - This network is used for providing separate connectivity between Swift Proxy and Storage Nodes. This ensures storage traffic is not interfering with Instance traffic.
  - This network typically employs private (RFC1918) IP addressing.

Figure 1 is used to help visualize the network deployment and to act as a reference for configuration steps within the document. It is highly recommend to print the diagram so it can easily be referenced throughout the installation process.

Figure 1:OpenStack HA Network Design Details

- **Other Network Services**
  - **DNS:** In this setup an external DNS server (192.168.26.186) is used for name resolution of OpenStack nodes and external name resolution. If DNS is not being used, the /etc/hosts file should include the following for all nodes:

```
127.0.0.1       localhost
192.168.220.40  control.dmz-pod2.lab           control
192.168.220.41  control01.dmz-pod2.lab         control01
192.168.220.42  control02.dmz-pod2.lab         control02
192.168.220.43  control03.dmz-pod2.lab         control03
192.168.220.60  swiftproxy.dmz-pod2.lab        swiftproxy
192.168.220.61  swiftproxy01.dmz-pod2.lab      swiftproxy01
192.168.220.62  swiftproxy02.dmz-pod2.lab      swiftproxy02
192.168.220.51  compute01.dmz-pod2.lab         compute01
192.168.220.52  compute02.dmz-pod2.lab         compute02
```

```
192.168.220.53  compute03.dmz-pod2.lab          compute03
```

- **NTP:** In this setup an external NTP server(s) is used for time synchronization.
- **Physical Network Switches:** Each node in this setup is physically attached to a Cisco Nexus switch acting as a Top-of-Rack access layer device. Trunking is configured on each interface connecting to the eth0 NIC of each node.

Note: Upstream routers/aggregation layer switches will most likely be terminating the Layer-3 VLAN interfaces. If these interfaces are deployed in a redundant fashion with a First Hop Redundancy Protocol such as HSRP or VRRP, then you should be careful of the IP addresses assigned to the physical L3 switches/routers as they may conflict with the IP address of the Quantum router's public subnet (.3 by default). For example, if you are using HSRP and you have .1 as the standby IP address, .2 as the first L3 switch IP and .3 as the second L3 switch IP, you will receive a duplicate IP address error on the second L3 switch. This can be worked around by using high-order IPs on your upstream L3 device or altering the Quantum subnet configuration at the time of creation to have an IP starting range higher than the physical switches/routers are using (i.e. .4 and higher). Our example uses an IP allocation range that starts with .10 to avoid this issue.

Installation

The installation of the nodes should be in the following order:

1. **Load-Balancer Nodes-** slb01 and slb02
2. **Swift Storage Nodes-** swift01, swift02 and swift03
3. **Swift Proxy Nodes-** swiftproxy01 and swiftproxy02
4. **Controller Nodes-** control01, control02 and control03
5. **Compute Nodes-** compute01, compute02 and compute03

General Installation Steps for All Nodes

Ubuntu Precise 12.04 Installation

Install Ubuntu 12.04 (AMD 64-bit) from CD/ISO or automated install (i.e. kickstart). You can reference Section 4 in the Build Node Guide if you are unfamiliar with the Ubuntu Precise installation process. Use the following networking section to configure your network adapter properties for each node. As previously mentioned in the Critical Reminders Section, make sure to create an LVM Volume Group named nova-volumes for Compute Nodes and do not configure RAID for Swift Storage Nodes. Lastly, select ssh-server as the only additional package during the Ubuntu Precise installation.

Grizzly Packages

Canonical?s Ubuntu Cloud Archive allows users the ability to install newer releases of OpenStack (and dependencies) on Ubuntu Server 12.04 LTS as they become available up through the next Ubuntu LTS release. Canonical commits to maintaining and supporting new OpenStack releases for Ubuntu Server 12.04 LTS in the Ubuntu Cloud archive for at least 18 months after they release. The Ubuntu Cloud Archive should be used for all OpenStack nodes (i.e. not needed for Load-Balancer nodes).

Use sudo mode or run from root account for the entire installation:

```
sudo su
```

Add the cloud archive gpg key into your ubuntu-keyring:

```
apt-get install ubuntu-cloud-keyring
```

Enable the Ubuntu Cloud Archive repository by adding the following to /etc/apt/sources.list.d/grizzly.list:

```
deb http://ubuntu-cloud.archive.canonical.com/ubuntu precise-updates/grizzly main
```

Update your system:

```
apt-get update && apt-get upgrade
```

Networking

Our implementation uses VLANs for segmentation of certain networks. Make sure the VLAN package is installed and your network switches have been configured for VLANs. Otherwise, replicate the network setup using only physical interfaces:

```
apt-get install vlan -y
```

Load-Balancer Node slb01 /etc/network/interfaces:

```
# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
auto eth0
iface eth0 inet static
        address 192.168.220.81
        netmask 255.255.255.0
        network 192.168.220.0
        broadcast 192.168.220.255
        gateway 192.168.220.1
        # dns-* options are implemented by the resolvconf package, if installed
        dns-nameservers 192.168.220.254
        dns-search dmz-pod2.lab
```

Load-Balancer Node slb02 /etc/network/interfaces:

```
# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
auto eth0
iface eth0 inet static
        address 192.168.220.82
        netmask 255.255.255.0
        network 192.168.220.0
        broadcast 192.168.220.255
        gateway 192.168.220.1
        # dns-* options are implemented by the resolvconf package, if installed
        dns-nameservers 192.168.220.254
        dns-search dmz-pod2.lab
```

Storage Node swift01 /etc/network/interfaces:

```
# The loopback network interface
```

```
auto lo
iface lo inet loopback

# Management Network
auto eth0
iface eth0 inet static
        address 192.168.220.71
        netmask 255.255.255.0
        network 192.168.220.0
        broadcast 192.168.220.255
        gateway 192.168.220.1
        dns-search dmz-pod2.lab
        dns-nameservers 192.168.220.254


# Storage Network
auto eth0.222
iface eth0.222 inet static
        address 192.168.222.71
        netmask 255.255.255.0
```

Storage Node swift02 /etc/network/interfaces:

```
# The loopback network interface
auto lo
iface lo inet loopback

# Management Network
auto eth0
iface eth0 inet static
        address 192.168.220.72
        netmask 255.255.255.0
        network 192.168.220.0
        broadcast 192.168.220.255
        gateway 192.168.220.1
        dns-search dmz-pod2.lab
        dns-nameservers 192.168.220.254


# Storage Network
auto eth0.222
iface eth0.222 inet static
        address 192.168.222.72
        netmask 255.255.255.0
```

Storage Node swift03 /etc/network/interfaces:

```
# The loopback network interface
auto lo
iface lo inet loopback

# Management Network
auto eth0
iface eth0 inet static
        address 192.168.220.73
        netmask 255.255.255.0
        network 192.168.220.0
        broadcast 192.168.220.255
        gateway 192.168.220.1
```

```
        dns-search dmz-pod2.lab
        dns-nameservers 192.168.220.254


# Storage Network
auto eth0.222
iface eth0.222 inet static
        address 192.168.222.73
        netmask 255.255.255.0
```

- Proxy Node swiftproxy01 /etc/network/interfaces:

```
# The loopback network interface
auto lo
iface lo inet loopback

# Management Network
auto eth0
iface eth0 inet static
        address 192.168.220.61
        netmask 255.255.255.0
        network 192.168.220.0
        broadcast 192.168.220.255
        gateway 192.168.220.1
        dns-search dmz-pod2.lab
        dns-nameservers 192.168.220.254

# Storage Network
auto eth0.222
iface eth0.222 inet static
        address 192.168.222.61
        netmask 255.255.255.0
```

Proxy Node swiftproxy02 /etc/network/interfaces:

```
# The loopback network interface
auto lo
iface lo inet loopback

# Management Network
auto eth0
iface eth0 inet static
        address 192.168.220.62
        network 192.168.220.0
        netmask 255.255.255.0
        broadcast 192.168.220.255
        gateway 192.168.220.1
        dns-search dmz-pod2.lab
        dns-nameservers 192.168.220.254

# Storage Network
auto eth0.222
iface eth0.222 inet static
        address 192.168.222.62
        netmask 255.255.255.0
```

Control Node control01 /etc/network/interfaces:

```
# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
auto eth0
iface eth0 inet static
        address 192.168.220.41
        netmask 255.255.255.0
        network 192.168.220.0
        broadcast 192.168.220.255
        gateway 192.168.220.1
        # dns-* options are implemented by the resolvconf package, if installed
        dns-nameservers 192.168.220.254
        dns-search dmz-pod2.lab
```

Control Node control02 /etc/network/interfaces:

```
# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
auto eth0
iface eth0 inet static
        address 192.168.220.42
        netmask 255.255.255.0
        network 192.168.220.0
        broadcast 192.168.220.255
        gateway 192.168.220.1
        # dns-* options are implemented by the resolvconf package, if installed
        dns-nameservers 192.168.220.254
        dns-search dmz-pod2.lab
```

Control Node control03 /etc/network/interfaces:

```
# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
auto eth0
iface eth0 inet static
        address 192.168.220.43
        netmask 255.255.255.0
        network 192.168.220.0
        broadcast 192.168.220.255
        gateway 192.168.220.1
        # dns-* options are implemented by the resolvconf package, if installed
        dns-nameservers 192.168.220.254
        dns-search dmz-pod2.lab
```

Compute Node compute01 /etc/network/interfaces:

```
# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
auto eth0
iface eth0 inet static
        address 192.168.220.51
```

```
        netmask 255.255.255.0
        network 192.168.220.0
        broadcast 192.168.220.255
        gateway 192.168.220.1
        # dns-* options are implemented by the resolvconf package, if installed
        dns-nameservers 192.168.220.254
        dns-search dmz-pod2.lab

# Public Network: Bridged Interface
auto eth1
iface eth1 inet manual
        up ifconfig $IFACE 0.0.0.0 up
        up ip link set $IFACE promisc on
        down ifconfig $IFACE 0.0.0.0 down
```

Compute Node compute02 /etc/network/interfaces:

```
# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
auto eth0
iface eth0 inet static
        address 192.168.220.52
        netmask 255.255.255.0
        network 192.168.220.0
        broadcast 192.168.220.255
        gateway 192.168.220.1
        # dns-* options are implemented by the resolvconf package, if installed
        dns-nameservers 192.168.220.254
        dns-search dmz-pod2.lab

# Public Network: Bridged Interface
auto eth1
iface eth1 inet manual
        up ifconfig $IFACE 0.0.0.0 up
        up ip link set $IFACE promisc on
        down ifconfig $IFACE 0.0.0.0 down
```

Compute Node compute03 /etc/network/interfaces:

```
# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
auto eth0
iface eth0 inet static
        address 192.168.220.53
        netmask 255.255.255.0
        network 192.168.220.0
        broadcast 192.168.220.255
        gateway 192.168.220.1
        # dns-* options are implemented by the resolvconf package, if installed
        dns-nameservers 192.168.220.254
        dns-search dmz-pod2.lab

# Public Network: Bridged Interface
auto eth1
iface eth1 inet manual
        up ifconfig $IFACE 0.0.0.0 up
        up ip link set $IFACE promisc on
```

```
        down ifconfig $IFACE 0.0.0.0 down
```

Restart networking:

```
/etc/init.d/networking restart
```

Time Synchronization

Install NTP:

```
apt-get install -y ntp
```

Add your NTP server(s) by editing /etc/ntp.conf. **Note:** OpenStack requires that clocks be synchronized. Our example uses a **FAKE** server called ntp.corp.com as the NTP server. Make sure you change ntp.corp.com to your real NTP server. Lastly, make sure the NTP server name resolves.

```
vi /etc/ntp.conf

server ntp.corp.com
```

Restart NTP for the changes to take effect

```
service ntp restart
```

Verify that you are pulling time:

```
ntpq -p

remote           refid      st t when poll reach   delay   offset  jitter
==============================================================================
*ntp.corp.       .GPS.            1 u  185  512  377   76.035    0.053   0.033
 cheezum.mattnor 129.7.1.66       2 u   8d 1024    0   47.731   -0.555   0.000
 ntp2.rescomp.be .STEP.          16 u    - 1024    0    0.000    0.000   0.000
 216.45.57.38    204.123.2.5      2 u  54h 1024    0   12.607    0.808   0.000
 lithium.constan 128.4.1.1        2 u   8d 1024    0   69.861    0.206   0.000
 europium.canoni 193.79.237.14    2 u  54h 1024    0  144.040   -1.455   0.000
```

Load Balancer Node Installation

Ensure you have completed the steps in the General Installation Steps for All Nodes section before proceeding. Perform the following steps on nodes slb01 and slb02.

Keepalived & HAProxy

Edit /etc/sysctl.conf to allow Keepalived to associate a virtual IP address (VIP) that is not directly bound to an interface on the node:

```
net.ipv4.ip_nonlocal_bind=1
```

Load in sysctl settings from /etc/sysctl.conf:

```
sysctl -p
```

Install Keepalived and HAProxy packages:

```
apt-get install -y keepalived haproxy
```

Create the /var/lib/haproxy directory:

```
mkdir /var/lib/haproxy
```

Make sure /var/lib/haproxy is owned by root. Change the file ownership if needed:

```
chown root:root /var/lib/haproxy/
```

Configure the /etc/keepalived/keepalived.conf file for slb01 with the contents below.
Change [YOUR_DOMAIN_NAME] with your actual domain name. The keepalived.conf includes the following sections:

- **global_defs-** Global parameters affect the whole process behavior. There may be several 'global' sections if needed, but their parameters will only be merged.
- **vrrp_script-** Keepalived supports a VRRP scripting framework to extend base functionality. The vrrp_script named haproxy will check the status of the haproxy service every 2 seconds and add 2 points of priority if the status is running. If the haproxy service is not running, the backup HAProxy Node will become the primary and begin passing traffic for the virtual_ipaddress(es).
- **vrrp_instance-** Is where you define configuration paramters for virtual gateway addresses. slb01 is configured as the primary gateway for 192.168.220.40 (Controller Cluster) and the backup gateway for 192.168.220.60 (Swift Proxy Cluster). Accordingly, slb02 is configured as the primary for 192.168.220.60 and the backup for 192.168.220.40.

```
global_defs {
  notification_email {
    root@[YOUR_DOMAIN_NAME]
  }
  notification_email_from keepalived@[YOUR_DOMAIN_NAME]
  smtp_server localhost
  smtp_connect_timeout 30
  router_id slb01
}
vrrp_script haproxy {
  script   "killall -0 haproxy"
  interval 2
  weight   2
}
vrrp_instance 50 {
  virtual_router_id 50

  # Advert interval
  advert_int 1

  # for electing MASTER, highest priority wins.
  priority  101
  state     MASTER
  interface eth0
  virtual_ipaddress {
      192.168.220.40 dev eth0
  }
  track_script {
    haproxy
  }
}
vrrp_instance 51 {
  virtual_router_id 51
```

```
  # Advert interval
  advert_int 1

  # for electing MASTER, highest priority wins.
  priority  100
  state     BACKUP
  interface eth0
  virtual_ipaddress {
      192.168.220.60 dev eth0
  }
  track_script {
    haproxy
  }
}
```

Configure /etc/keepalived/keepalived.conf for slb02 with the following contents. Change [YOUR_DOMAIN_NAME] with your actual domain name.

```
global_defs {
  notification_email {
    root@[YOUR_DOMAIN_NAME]
  }
  notification_email_from keepalived@[YOUR_DOMAIN_NAME]
  smtp_server localhost
  smtp_connect_timeout 30
  router_id slb02
}
vrrp_script haproxy {
  script   "killall -0 haproxy"
  interval 2
  weight   2
}
vrrp_instance 50 {
  virtual_router_id 50
  # Advert interval
  advert_int 1
  # for electing MASTER, highest priority wins.
  priority  100
  state     BACKUP
  interface eth0
  virtual_ipaddress {
      192.168.220.40 dev eth0
  }
  track_script {
    haproxy
  }
}
vrrp_instance 51 {
  virtual_router_id 51
  # Advert interval
  advert_int 1
  # for electing MASTER, highest priority wins.
  priority  101
  state     MASTER
  interface eth0
  virtual_ipaddress {
      192.168.220.60 dev eth0
  }
  track_script {
    haproxy
  }
}
```

Configure the /etc/haproxy/haproxy.cfg file for slb01 with the contents below. HAProxy's configuration process involves 3 major sources of parameters:

- The arguments from the command-line, which always take precedence.
- The "global" section, which sets process-wide parameters.
- The proxies sections which can take form of "defaults", "listen", "frontend" and "backend".

The following provides additional details of the haproxy.cfg file:

- **global-** Sets process-wide parameters for load-balancing traffic. Global parameters can be overriden by server-specific configurations within the*listen section* of the haproxy.cfg file.
- **defaults-** The "defaults" section sets default parameters for all other sections following its declaration. Those default parameters are reset by the next "defaults" section. The name is optional but its use is encouraged for better readability.
- **listen-** A "listen" section defines a complete proxy with its front-end (i.e. listening VIP) and back-end (i.e. real IP of servers) parts combined in one section. Currently two major proxy modes are supported: "tcp", also known as layer 4 and "http", also known as layer 7. In layer 4 mode, HAProxy simply forwards bidirectional traffic between two sides. In layer 7 mode, HAProxy analyzes the protocol and can interact with it by allowing, blocking, switching, adding, modifying, or removing arbitrary content in requests or responses based on configurable criteria.

```
global
 chroot  /var/lib/haproxy
 daemon
 group  haproxy
 log  192.168.220.81 local0
 maxconn  4000
 pidfile  /var/run/haproxy.pid
 user  haproxy

defaults
 log  global
 maxconn  8000
 option  redispatch
 retries  3
 timeout  http-request 10s
 timeout  queue 1m
 timeout  connect 10s
 timeout  client 1m
 timeout  server 1m
 timeout  check 10s

listen dashboard_cluster
 bind 192.168.220.40:80
 balance  source
 option  tcpka
 option  httpchk
 option  tcplog
 server control01 192.168.220.41:80 check inter 2000 rise 2 fall 5
 server control02 192.168.220.42:80 check inter 2000 rise 2 fall 5
 server control03 192.168.220.43:80 check inter 2000 rise 2 fall 5

listen galera_cluster
 bind 192.168.220.40:3306
 balance  source
 mode  tcp
 option  httpchk
 server control01 192.168.220.41:3306 check port 9200 inter 2000 rise 2 fall 5
 server control02 192.168.220.42:3306 check port 9200 inter 2000 rise 2 fall 5
 server control03 192.168.220.43:3306 check port 9200 inter 2000 rise 2 fall 5
```

```
listen glance_api_cluster
 bind 192.168.220.40:9292
 balance  source
 option  tcpka
 option  httpchk
 option  tcplog
 server control01 192.168.220.41:9292 check inter 2000 rise 2 fall 5
 server control02 192.168.220.42:9292 check inter 2000 rise 2 fall 5
 server control03 192.168.220.43:9292 check inter 2000 rise 2 fall 5

listen glance_registry_cluster
 bind 192.168.220.40:9191
 balance  source
 option  tcpka
 option  tcplog
 server control01 192.168.220.41:9191 check inter 2000 rise 2 fall 5
 server control02 192.168.220.42:9191 check inter 2000 rise 2 fall 5
 server control03 192.168.220.43:9191 check inter 2000 rise 2 fall 5

listen keystone_admin_cluster
 bind 192.168.220.40:35357
 balance  source
 option  tcpka
 option  httpchk
 option  tcplog
 server control01 192.168.220.41:35357 check inter 2000 rise 2 fall 5
 server control02 192.168.220.42:35357 check inter 2000 rise 2 fall 5
 server control03 192.168.220.43:35357 check inter 2000 rise 2 fall 5

listen keystone_public_internal_cluster
 bind 192.168.220.40:5000
 balance  source
 option  tcpka
 option  httpchk
 option  tcplog
 server control01 192.168.220.41:5000 check inter 2000 rise 2 fall 5
 server control02 192.168.220.42:5000 check inter 2000 rise 2 fall 5
 server control03 192.168.220.43:5000 check inter 2000 rise 2 fall 5

listen memcached_cluster
 bind 192.168.220.40:11211
 balance  source
 option  tcpka
 option  tcplog
 server control01 192.168.220.41:11211 check inter 2000 rise 2 fall 5
 server control02 192.168.220.42:11211 check inter 2000 rise 2 fall 5
 server control03 192.168.220.43:11211 check inter 2000 rise 2 fall 5

listen nova_compute_api1_cluster
 bind 192.168.220.40:8773
 balance  source
 option  tcpka
 option  tcplog
 server control01 192.168.220.41:8773 check inter 2000 rise 2 fall 5
 server control02 192.168.220.42:8773 check inter 2000 rise 2 fall 5
 server control03 192.168.220.43:8773 check inter 2000 rise 2 fall 5

listen nova_compute_api2_cluster
 bind 192.168.220.40:8774
 balance  source
 option  tcpka
 option  httpchk
 option  tcplog
```

```
 server control01 192.168.220.41:8774 check inter 2000 rise 2 fall 5
 server control02 192.168.220.42:8774 check inter 2000 rise 2 fall 5
 server control03 192.168.220.43:8774 check inter 2000 rise 2 fall 5

listen nova_compute_api3_cluster
 bind 192.168.220.40:8775
 balance  source
 option  tcpka
 option  tcplog
 server control01 192.168.220.41:8775 check inter 2000 rise 2 fall 5
 server control02 192.168.220.42:8775 check inter 2000 rise 2 fall 5
 server control03 192.168.220.43:8775 check inter 2000 rise 2 fall 5

listen nova_volume_cluster
 bind 192.168.220.40:8776
 balance  source
 option  tcpka
 option  httpchk
 option  tcplog
 server control01 192.168.220.41:8776 check inter 2000 rise 2 fall 5
 server control02 192.168.220.42:8776 check inter 2000 rise 2 fall 5
 server control03 192.168.220.43:8776 check inter 2000 rise 2 fall 5

listen novnc_cluster
 bind 192.168.220.40:6080
 balance  source
 option  tcpka
 option  tcplog
 server control01 192.168.220.41:6080 check inter 2000 rise 2 fall 5
 server control02 192.168.220.42:6080 check inter 2000 rise 2 fall 5
 server control03 192.168.220.43:6080 check inter 2000 rise 2 fall 5

listen quantum_api_cluster
 bind 192.168.220.40:9696
 balance  source
 option  tcpka
 option  httpchk
 option  tcplog
 server control01 192.168.220.41:9696 check inter 2000 rise 2 fall 5
 server control02 192.168.220.42:9696 check inter 2000 rise 2 fall 5
 server control03 192.168.220.43:9696 check inter 2000 rise 2 fall 5

listen swift_proxy_cluster
 bind 192.168.220.60:8080
 balance  source
 option  tcplog
 option  tcpka
 server swiftproxy01 192.168.220.61:8080 check inter 2000 rise 2 fall 5
 server swiftproxy02 192.168.220.62:8080 check inter 2000 rise 2 fall 5
```

Configure the /etc/haproxy/haproxy.cfg file for slb02 with the contents below.

```
global
  chroot  /var/lib/haproxy
  daemon
  group  haproxy
  log  192.168.220.82 local0
  maxconn  4000
  pidfile  /var/run/haproxy.pid
  user  haproxy

defaults
  log  global
```

```
  maxconn  8000
  option  redispatch
  retries  3
  timeout  http-request 10s
  timeout  queue 1m
  timeout  connect 10s
  timeout  client 1m
  timeout  server 1m
  timeout  check 10s

listen dashboard_cluster
  bind 192.168.220.40:80
  balance  source
  option  tcpka
  option  httpchk
  option  tcplog
  server control01 192.168.220.41:80 check inter 2000 rise 2 fall 5
  server control02 192.168.220.42:80 check inter 2000 rise 2 fall 5
  server control03 192.168.220.43:80 check inter 2000 rise 2 fall 5

listen galera_cluster
  bind 192.168.220.40:3306
  balance  source
  option  httpchk
  server control01 192.168.220.41:3306 check port 9200 inter 2000 rise 2 fall 5
  server control02 192.168.220.42:3306 check port 9200 inter 2000 rise 2 fall 5
  server control03 192.168.220.43:3306 check port 9200 inter 2000 rise 2 fall 5

listen glance_api_cluster
  bind 192.168.220.40:9292
  balance  source
  option  tcpka
  option  httpchk
  option  tcplog
  server control01 192.168.220.41:9292 check inter 2000 rise 2 fall 5
  server control02 192.168.220.42:9292 check inter 2000 rise 2 fall 5
  server control03 192.168.220.43:9292 check inter 2000 rise 2 fall 5

listen glance_registry_cluster
  bind 192.168.220.40:9191
  balance  source
  option  tcpka
  option  tcplog
  server control01 192.168.220.41:9191 check inter 2000 rise 2 fall 5
  server control02 192.168.220.42:9191 check inter 2000 rise 2 fall 5
  server control03 192.168.220.43:9191 check inter 2000 rise 2 fall 5

listen keystone_admin_cluster
  bind 192.168.220.40:35357
  balance  source
  option  tcpka
  option  httpchk
  option  tcplog
  server control01 192.168.220.41:35357 check inter 2000 rise 2 fall 5
  server control02 192.168.220.42:35357 check inter 2000 rise 2 fall 5
  server control03 192.168.220.43:35357 check inter 2000 rise 2 fall 5

listen keystone_public_internal_cluster
  bind 192.168.220.40:5000
  balance  source
  option  tcpka
  option  httpchk
  option  tcplog
  server control01 192.168.220.41:5000 check inter 2000 rise 2 fall 5
```

```
  server control02 192.168.220.42:5000 check inter 2000 rise 2 fall 5
  server control03 192.168.220.43:5000 check inter 2000 rise 2 fall 5

listen memcached_cluster
  bind 192.168.220.40:11211
  balance  source
  option  tcpka
  option  tcplog
  server control01 192.168.220.41:11211 check inter 2000 rise 2 fall 5
  server control02 192.168.220.42:11211 check inter 2000 rise 2 fall 5
  server control03 192.168.220.43:11211 check inter 2000 rise 2 fall 5

listen nova_compute_api1_cluster
  bind 192.168.220.40:8773
  balance  source
  option  tcpka
  option  tcplog
  server control01 192.168.220.41:8773 check inter 2000 rise 2 fall 5
  server control02 192.168.220.42:8773 check inter 2000 rise 2 fall 5
  server control03 192.168.220.43:8773 check inter 2000 rise 2 fall 5

listen nova_compute_api2_cluster
  bind 192.168.220.40:8774
  balance  source
  option  tcpka
  option  httpchk
  option  tcplog
  server control01 192.168.220.41:8774 check inter 2000 rise 2 fall 5
  server control02 192.168.220.42:8774 check inter 2000 rise 2 fall 5
  server control03 192.168.220.43:8774 check inter 2000 rise 2 fall 5

listen nova_compute_api3_cluster
  bind 192.168.220.40:8775
  balance  source
  option  tcpka
  option  tcplog
  server control01 192.168.220.41:8775 check inter 2000 rise 2 fall 5
  server control02 192.168.220.42:8775 check inter 2000 rise 2 fall 5
  server control03 192.168.220.43:8775 check inter 2000 rise 2 fall 5

listen nova_volume_cluster
  bind 192.168.220.40:8776
  balance  source
  option  tcpka
  option  httpchk
  option  tcplog
  server control01 192.168.220.41:8776 check inter 2000 rise 2 fall 5
  server control02 192.168.220.42:8776 check inter 2000 rise 2 fall 5
  server control03 192.168.220.43:8776 check inter 2000 rise 2 fall 5

listen novnc_cluster
  bind 192.168.220.40:6080
  balance  source
  option  tcpka
  option  tcplog
  server control01 192.168.220.41:6080 check inter 2000 rise 2 fall 5
  server control02 192.168.220.42:6080 check inter 2000 rise 2 fall 5
  server control03 192.168.220.43:6080 check inter 2000 rise 2 fall 5

listen quantum_api_cluster
  bind 192.168.220.40:9696
  balance  source
  option  tcpka
  option  httpchk
```

```
  option  tcplog
  server control01 192.168.220.41:9696 check inter 2000 rise 2 fall 5
  server control02 192.168.220.42:9696 check inter 2000 rise 2 fall 5
  server control03 192.168.220.43:9696 check inter 2000 rise 2 fall 5

listen swift_proxy_cluster
  bind 192.168.220.60:8080
  balance  source
  option  tcplog
  option  tcpka
  server swiftproxy01 192.168.220.61:8080 check inter 2000 rise 2 fall 5
  server swiftproxy02 192.168.220.62:8080 check inter 2000 rise 2 fall 5
```

Set "ENABLED" to "1" in /etc/default/haproxy

```
ENABLED=1
```

Restart Keepalived and HAProxy services:

```
/etc/init.d/keepalived restart
/etc/init.d/haproxy restart
```

General Installation Steps for All Swift Nodes

Make sure to complete the General Installation Steps for All Nodes section before proceeding. Install Swift and other basic packages:

```
apt-get install -y swift openssh-server rsync memcached python-netifaces python-xattr python-memca
```

Create the Swift configuration directory:

```
mkdir -p /etc/swift
```

Create the Swift configuration file. **Note:** This file should be identical on all Swift nodes.

```
vi /etc/swift/swift.conf

[swift-hash]
swift_hash_path_suffix = Gdr8ny7YyWqy2
```

Change the ownership of the Swift directory:

```
chown -R swift:swift /etc/swift/
```

Swift Storage Node Installation Steps

Ensure you have completed the steps in the General Installation Steps for All Nodes and General Installation Steps for All Swift Nodes sections before proceeding. Run these commands on nodes swift01, swift02 and swift03. Install the Swift Storage Node packages:

```
apt-get install -y swift-account swift-container swift-object xfsprogs parted
```

For each of the hard disks other than the Ubuntu installation disk (i.e. /dev/sda), create an XFS volume with a single partition. Our example uses five hard disks (/dev/sdb - /dev/sdf) per Storage Node. Repeat this step for

each disk that will be used for Swift storage. **Note:** You can ignore the following message when using the parted commands*"Information: You may need to update /etc/fstab"*

```
parted /dev/sdb mklabel msdos
parted -a optimal /dev/sdb mkpart primary ext2 0% 100%
mkfs.xfs -i size=1024 /dev/sdb1
echo "/dev/sdb1 /srv/node/sdb1 xfs noatime,nodiratime,nobarrier,logbufs=8 0 0" >> /etc/fstab
mkdir -p /srv/node/sdb1
mount /srv/node/sdb1
```

Change the ownership of the mount directory:

```
chown -R swift:swift /srv/node
```

Create an Rsync configuration file on each Storage Node. In the following example, replace [STORAGE_NET_IP] with the node's storage network IP address (i.e. swift01 = 192.168.222.71):

```
vi /etc/rsyncd.conf

uid = swift
gid = swift
log file = /var/log/rsyncd.log
pid file = /var/run/rsyncd.pid
address = [STORAGE_NET_IP]

[account]
max connections = 2
path = /srv/node/
read only = false
lock file = /var/lock/account.lock

[container]
max connections = 2
path = /srv/node/
read only = false
lock file = /var/lock/container.lock

[object]
max connections = 2
path = /srv/node/
read only = false
lock file = /var/lock/object.lock
```

Edit the following line in /etc/default/rsync:

```
RSYNC_ENABLE = true
```

Start rsync daemon:

```
service rsync start
```

Edit /etc/swift/account-server.conf with the following contents. Replace [STORAGE_NET_IP] with the node's storage network IP address (i.e. swift01 = 192.168.222.71):

```
vi /etc/swift/account-server.conf

[DEFAULT]
bind_ip = [STORAGE_NET_IP]
workers = 2
```

```
[pipeline:main]
pipeline = account-server

[app:account-server]
use = egg:swift#account

[account-replicator]

[account-auditor]

[account-reaper]
```

Edit /etc/swift/container-server.conf with the following contents. Replace [STORAGE_NET_IP] with the node's storage network IP address (i.e. swift01 = 192.168.222.71):

```
vi /etc/swift/container-server.conf

[DEFAULT]
bind_ip = [STORAGE_NET_IP]
workers = 2

[pipeline:main]
pipeline = container-server

[app:container-server]
use = egg:swift#container

[container-replicator]

[container-updater]

[container-auditor]
```

Create /etc/swift/object-server.conf with the following contents. Replace [STORAGE_NET_IP] with the node's storage network IP address (i.e. swift01 = 192.168.222.71):

```
vi /etc/swift/object-server.conf

[DEFAULT]
bind_ip = [STORAGE_NET_IP]
workers = 2

[pipeline:main]
pipeline = object-server

[app:object-server]
use = egg:swift#object

[object-replicator]

[object-updater]

[object-auditor]

[object-expirer]
```

Start the storage services.

Note: At this point, the ring files will not be present on the storage nodes. This will cause the *-replicator services to fail to start properly. After you create the ring files on the first proxy node (in the next section) and distribute them to the storage nodes, a service restart should allow all Swift services to start properly.

```
swift-init object-server start
swift-init object-replicator start
swift-init object-updater start
swift-init object-auditor start
swift-init container-server start
swift-init container-replicator start
swift-init container-updater start
swift-init container-auditor start
swift-init account-server start
swift-init account-replicator start
swift-init account-auditor start
```

Make sure you repeat these steps for every Storage Node.

Swift Proxy Node Installation Steps

Ensure you have completed the steps in the General Installation Steps for All Nodes and General Installation Steps for All Swift Nodes sections before proceeding. Perform these steps on nodes swiftproxy01 and swiftproxy02. First, install the proxy node packages:

```
apt-get install -y swift-proxy memcached python-keystoneclient python-swiftclient swift-plugin-s3
```

Modify memcached to bind to the storage network interface (192.168.222.x in our example). Edit the following line in /etc/memcached.conf, changing:

```
-l 127.0.0.1
to
-l [STORAGE_NET_IP]
```

Restart the memcached server:

```
service memcached restart
```

If it does not exist, create the /etc/swift/ directory:

```
mkdir /etc/swift/
```

If /etc/swift and /var/cache/swift directories are not owned by the swift user and group, then change the ownership of the directories:

```
chown -R swift:swift /etc/swift/
chown -R swift:swift /var/cache/swift/
```

Create /etc/swift/proxy-server.conf with the following contents:

```
[DEFAULT]
bind_port = 8080
workers = 32
user = swift

[pipeline:main]
pipeline = catch_errors healthcheck cache ratelimit authtoken keystoneauth proxy-server

[app:proxy-server]
use = egg:swift#proxy
allow_account_management = true
account_autocreate = true
```

```
[filter:keystoneauth]
use = egg:swift#keystoneauth
operator_roles = Member,admin, swiftoperator

[filter:authtoken]
paste.filter_factory = keystoneclient.middleware.auth_token:filter_factory
signing_dir = /var/cache/swift
auth_host = 192.168.220.40
auth_port = 35357
auth_protocol = http
auth_uri = http://192.168.220.40:5000
admin_tenant_name = services
admin_user = swift
admin_password = keystone_admin
delay_auth_decision = 10

[filter:cache]
use = egg:swift#memcache
memcache_servers = 192.168.222.61:11211,192.168.222.62:11211

[filter:catch_errors]
use = egg:swift#catch_errors

[filter:healthcheck]
use = egg:swift#healthcheck

[filter:ratelimit]
use = egg:swift#ratelimit
clock_accuracy = 1000
max_sleep_time_seconds = 60
log_sleep_time_seconds = 0
rate_buffer_seconds = 5
account_ratelimit = 0
```

On all proxy nodes, create the account, container and object rings. The builder command is basically creating a builder file with a few parameters. The parameter with the value of 18 represents 2 ^ 18th, this is the value of the partition size. Set this ?partition power? value based on the total amount of storage you expect your entire ring to use. The value of 3 represents the number of replicas of each object, with the last value being the number of hours to restrict moving a partition more than once. Additional information regarding Swift ring preparation can be found here.

```
cd /etc/swift

swift-ring-builder account.builder create 18 3 1
swift-ring-builder container.builder create 18 3 1
swift-ring-builder object.builder create 18 3 1
```

On all proxy nodes, for every storage device on each storage node add entries to each ring. This example prepares the account, container and object rings for storage node swift01 (192.168.222.71) with a partition in zone 1. The mount point of this partition is /srv/node/sdb1 and the path in rsyncd.conf is /srv/node/, the DEVICE would be sdb1 and the commands would look like:

```
swift-ring-builder account.builder add z1-192.168.222.71:6002/sdb1 100
swift-ring-builder container.builder add z1-192.168.222.71:6001/sdb1 100
swift-ring-builder object.builder add z1-192.168.222.71:6000/sdb1 100
```

Note: Make sure not to place all devices in the same zone (i.e. z1). It is recommended to configure the zones as high-level as possible to create the greatest amount of isolation. Some considerations can include physical location, power availability, and network connectivity. For example, in a small cluster you might decide to split the zones up by cabinet, with each cabinet having its own power and network connectivity. Since our

deployment only uses 3 storage nodes, each node should be in its own zone. However, it is recommended to have a minimum of 5 zones in a production-level Swift deployment.

On all proxy nodes, verify the ring contents for each ring:

```
swift-ring-builder /etc/swift/account.builder
swift-ring-builder /etc/swift/container.builder
swift-ring-builder /etc/swift/object.builder
```

Your output should look similar to this:

```
root@swiftproxy01:~# swift-ring-builder /etc/swift/account.builder
/etc/swift/account.builder, build version 15
262144 partitions, 3 replicas, 3 zones, 15 devices, 0.00 balance
The minimum number of hours before a partition can be reassigned is 1
Devices:    id  zone      ip address  port      name weight partitions balance meta
             0     3  192.168.222.73  6002      sde1   1.00      52429    0.00
             1     2  192.168.222.72  6002      sdd1   1.00      52429    0.00
             2     3  192.168.222.73  6002      sdc1   1.00      52429    0.00
             3     2  192.168.222.72  6002      sdb1   1.00      52429    0.00
             4     3  192.168.222.73  6002      sdb1   1.00      52429    0.00
             5     1  192.168.222.71  6002      sdb1   1.00      52429    0.00
             6     1  192.168.222.71  6002      sdc1   1.00      52429    0.00
             7     2  192.168.222.72  6002      sdf1   1.00      52429    0.00
             8     1  192.168.222.71  6002      sdd1   1.00      52428   -0.00
             9     2  192.168.222.72  6002      sdc1   1.00      52429    0.00
            10     1  192.168.222.71  6002      sde1   1.00      52429    0.00
            11     1  192.168.222.71  6002      sdf1   1.00      52429    0.00
            12     3  192.168.222.73  6002      sdf1   1.00      52429    0.00
            13     2  192.168.222.72  6002      sde1   1.00      52428   -0.00
            14     3  192.168.222.73  6002      sdd1   1.00      52428   -0.00
```

On swiftproxy01, rebalance the rings. **Note:** Rebalancing rings can take a while. You may get a message about a balance value and that you need to rebalance/push after the minimum 1 hour. If so, recheck the status after an hour.

```
swift-ring-builder account.builder rebalance
swift-ring-builder container.builder rebalance
swift-ring-builder object.builder rebalance
```

On swiftproxy01, copy the account.ring.gz, container.ring.gz, and object.ring.gz files to swiftproxy02 and the 3 storage nodes in /etc/swift. Also make sure all the files in /etc/swift on all Swift nodes are owned by the swift user:

```
chown -R swift:swift /etc/swift
```

Start the Proxy services:

```
swift-init proxy start
```

REMINDER: After you have copied over the ring files and successfully restarted the proxy service, make sure you restart all Swift Storage Node services in the Swift Storage Node section of the document.

Verify the Swift Installation

You can run verification commands from the proxy server or any server with access to Keystone. Keep in mind that proxy nodes are configured to use Keystone for user authentication. As a result, you **MUST**

complete the Controller Node Installation steps and ensure Keystone is operational before proceeding with Swift verification.

Verify that you can successfully authenticate against Keystone using the Swift authentication credentials:

```
apt-get install -y curl

curl -s -d "{\"auth\":{\"passwordCredentials\": {\"username\": \"swift\", \"password\": \"keystone
```

You should receive output similar to the following:

```
{"access": {"token": {"issued_at": "2013-04-02T14:55:31.149327", "expires": "2013-04-03T14:55:31Z"
null, "enabled": true, "id": "b38d88aad6314870b746e7d60808e59a", "name": "services"}}, "serviceCat
"http://192.168.220.40:8774/v2/b38d88aad6314870b746e7d60808e59a", "region": "RegionOne", "internal
, "id": "45a336cb74e04e11ab95c0ea28b699d6", "publicURL": "http://192.168.220.40:8774/v2/b38d88aad6
"name": "nova"}, {"endpoints": [{"adminURL": "http://192.168.220.40:9696/", "region": "RegionOne",
"259fef5e66814f47ac1934d3cf522a3d", "publicURL": "http://192.168.220.40:9696/"}], "endpoints_links
{"adminURL": "http://192.168.220.40:9292/v2", "region": "RegionOne", "internalURL": "http://192.16
"publicURL": "http://192.168.220.40:9292/v2"}], "endpoints_links": [], "type": "image", "name": "g
"http://192.168.220.40:8776/v1/b38d88aad6314870b746e7d60808e59a", "region": "RegionOne", "internal
, "id": "0a2c69157d5948a9ae8ecee5c65a6d2b", "publicURL": "http://192.168.220.40:8776/v1/b38d88aad6
"name": "cinder"}, {"endpoints": [{"adminURL": "http://192.168.220.40:8773/services/Admin", "regio
"internalURL": "http://192.168.220.40:8773/services/Cloud", "id": "05f85b8aacbd4c87b680dcc2fb6da53
"endpoints_links": [], "type": "ec2", "name": "ec2"}, {"endpoints": [{"adminURL": "http://192.168.
"http://192.168.220.60:8080/v1/AUTH_b38d88aad6314870b746e7d60808e59a", "id": "4a1af526137341c0a682
"http://192.168.220.60:8080/v1/AUTH_b38d88aad6314870b746e7d60808e59a"}], "endpoints_links": [], "t
"http://192.168.220.40:35357/v2.0", "region": "RegionOne", "internalURL": "http://192.168.220.40:5
"http://192.168.220.40:5000/v2.0"}], "endpoints_links": [], "type": "identity", "name": "keystone"
"ed69664ac78a4b65a36d63da6b760863", "roles": [{"name": "_member_"}, {"name": "admin"}], "name": "s
"9fe2ff9ee4384b1894a90878d3e92bab", "6a553ae3be3c4f8c8fe079830d4102a5"]}}}
```

Use the swift client stat command to make sure you can view the contents of the ring. You can run these commands from the proxy server or any server with the swift client and access to Keystone.

```
swift -V 2 -A http://192.168.220.40:5000/v2.0/ -V 2 -U services:swift -K keystone_admin stat
   Account: AUTH_3eccdb2a9331419c96ac9ff336110b65
Containers: 1
   Objects: 2
     Bytes: 0
Accept-Ranges: bytes
X-Timestamp: 1363989109.30329
X-Trans-Id: tx147dd9983ac54af1b71c5a561ae2aa9a
Content-Type: text/plain; charset=utf-8
```

You can see that 1 container exists. Now, lets find out the name of the container:

```
swift -V 2 -A http://192.168.220.40:5000/v2.0/ -V 2 -U services:swift -K keystone_admin list
glance
```

Note: The glance container is created after the Controller cluster is built and an image has been uploaded to Glance.

List the contents of the Glance container:

```
swift -V 2 -A http://192.168.220.40:5000/v2.0/ -V 2 -U services:swift -K keystone_admin list glanc
24164630-ba2f-436a-8bc6-43975717d5e5
858a11dc-ed61-4a18-a778-eabcb454ae45
```

Controller Node Installation

Ensure you have completed the steps in the General Installation Steps for All Nodes section before proceeding. Run the following commands on nodes control01, control02 and control03.

MySQL WSREP and Galera Installation

Install MySQL and Galera dependencies:

```
apt-get install -y libaio1 libssl0.9.8 mysql-client-5.5 python-mysqldb
```

Download MySQL-WSREP and Galera:

```
wget -O /tmp/mysql-server-wsrep-5.5.23-23.6-amd64.deb http://launchpad.net/codership-mysql/5.5/5.5
wget -O /tmp/galera-23.2.1-amd64.deb http://launchpad.net/galera/2.x/23.2.1/+download/galera-23.2.
```

Install MySQL and Galera. **Note:** If you are prompted to create a root password during the Galera package installation, please make note of the password you use as it will be needed when connecting to MySQL:

```
dpkg -i /tmp/mysql-server-wsrep-5.5.23-23.6-amd64.deb
dpkg -i /tmp/galera-23.2.1-amd64.deb
```

Change the default MySQL bind address. Change [CONTROLLER_MGT_IP] in the command to the management IP address for each controller (i.e. control01 = 192.168.220.41):

```
sed -i 's/127.0.0.1/[CONTROLLER_MGT_IP]/g' /etc/mysql/my.cnf
```

Add the following line to /etc/rc.local on all controllers to allow MySQL to start automatically upon reboot:

```
service mysql start
```

Modify the default /etc/mysql/conf.d/wsrep.cnf file for control01:

```
bind-address=192.168.220.41
wsrep_provider=/usr/lib/galera/libgalera_smm.so
wsrep_cluster_name="controller_cluster"
wsrep_cluster_address="gcomm://"
wsrep_sst_method=rsync
wsrep_sst_auth=wsrep_sst:password
```

Modify the default /etc/mysql/conf.d/wsrep.cnf file for control02:

```
bind-address=192.168.220.42
wsrep_provider=/usr/lib/galera/libgalera_smm.so
wsrep_cluster_name="controller_cluster"
wsrep_cluster_address="gcomm://192.168.220.41"
wsrep_sst_method=rsync
wsrep_sst_auth=wsrep_sst:password
```

Modify the default /etc/mysql/conf.d/wsrep.cnf file for control03:

```
bind-address=192.168.220.43
wsrep_provider=/usr/lib/galera/libgalera_smm.so
wsrep_cluster_name="controller_cluster"
wsrep_cluster_address="gcomm://192.168.220.42"
wsrep_sst_method=rsync
wsrep_sst_auth=wsrep_sst:password
```

Note: It is important to understand the gcomm address concept behind Galera. Only use an empty gcomm:// address when you create a NEW cluster. Never use it when your intention is to reconnect to an existing one. After the Galera cluster is established, you should change the gcomm address on control01 from gcomm:// to gcomm://192.168.220.42 or gcomm://192.168.220.43. Otherwise, control01 will not join the cluster upon reboot. Make sure to also restart the mysql service when making changes to any of the associated configuration files. It is equally important to understand how to restart an existing Galera cluster in the event of a power outage.

Restart MySQL in the following order: control01, control02 and control03:

```
service mysql restart
```

Verify the Galera cluster has been established. The value should show 4 for all nodes in the cluster:

```
mysql -e "show global status where variable_name='wsrep_local_state';"
  +------------------------+---+
  | Variable_name      | Value |
  +------------------------+---+
  | wsrep_local_state  |    4  |
  +------------------------+---+
```

MySQL WSREP and Galera Monitoring

Complete each of the steps below on each control node except for when a single node is specified.

Install xinetd:

```
apt-get install -y xinetd
```

Download the mysqlchk service:

```
wget https://raw.github.com/CiscoSystems/puppet-mysql/folsom_ha/templates/mysqlchk -P /etc/xinetd.
```

Note: After functional testing is complete, it's recommended to secure the mysqlchk service. This can be accomplished by editing the only_from and per_source values in /etc/xinetd.d/ to the subnet used by the load-balancer nodes.

Edit /etc/xinetd.d/mysqlchk by changing <%= mysqlchk_script_dir %>/galera_chk to the following:

```
/usr/local/bin/galera_chk
```

Make sure root is the file owner:

```
ls -l /etc/xinetd.d/mysqlchk
```

If not, change the file permissions:

```
chown root:root /etc/xinetd.d/mysqlchk
```

Add the mysqlcheck service to /etc/services by adding the following line:

```
mysqlchk        9200/tcp                        # MySQL Galera health check script
```

Download the MySQL Galera health check script:

```
wget https://raw.github.com/CiscoSystems/puppet-mysql/folsom_ha/templates/galera_chk -P /usr/local
```

Set the file to be executable:

```
chmod +x  /usr/local/bin/galera_chk
```

Edit /usr/local/bin/galera_chk as follows. Change [CONTROLLER_MGT_IP] to the Management IP address for each controller node (i.e. control01 = 192.168.220.41).

```
MYSQL_HOST="[CONTROLLER_MGT_IP]"
MYSQL_PORT="3306"
MYSQL_USERNAME="mysqlchk_user"
MYSQL_PASSWORD="mysqlchk_password"
MYSQL_OPTS="-N -q -A"
TMP_FILE="/dev/shm/mysqlchk.$$.out"
ERR_FILE="/dev/shm/mysqlchk.$$.err"
FORCE_FAIL="/dev/shm/proxyoff"
MYSQL_BIN="/usr/bin/mysql"
```

Restart xinetsd:

```
service xinetd restart
```

Connect to MySQL and add the mysqlchk user to each controller in the cluster:

```
mysql
use mysql;
INSERT INTO user (Host,User,Password) VALUES('%','mysqlchk_user',PASSWORD('mysqlchk_password'));
flush privileges;
```

Grant privileges for the mysqlchk user. Change [CONTROLLER_MGT_IP] to the Management IP address for each controller node (i.e. control01 = 192.168.220.41).:

```
grant SUPER,PROCESS on *.* to 'mysqlchk_user'@'[CONTROLLER_MGT_IP]' IDENTIFIED BY 'mysqlchk_passwo
quit;
```

Verify the operational status of the MySQL Galera health check service. From slb01 or slb02, Telnet using port 9200 (health check port) and make sure you get a "MySQL is running" message:

```
telnet 192.168.220.41 9200
Trying 192.168.220.41...
Connected to 192.168.220.41.
Escape character is '^]'.
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 43
<html><body>MySQL is running.</body></html>
Connection closed by foreign host.
```

Repeat the previous step for each control node.

Verify that you can access the MySQL database by using the Virtual IP address (VIP) of the Galera cluster:

```
mysql -umysqlchk_user -pmysqlchk_password -h192.168.220.40
```

For informational purposes, this is the command used by the health check script. This example is for control01:

```
/usr/bin/mysql -N -q -A --host=192.168.220.41 --user=mysqlchk_user --password=mysqlchk_password -e
```

Upgrade Client Libraries

Several client libraries must be upgraded to support RabbitMQ Mirrored Queues.

Update the required client libraries:

```
apt-get install -y python-pip
pip install kombu==2.4.7
pip install amqp==0.9.4
```

Check your version of anyjson:

```
pip freeze | grep anyjson
```

If anyjson is not 0.3.3, then install the correct version:

```
pip install anyjson==0.3.3
```

RabbitMQ Installation

Complete the following steps on each control node unless a specific node is referenced.

A newer version of RabbitMQ Server is required for the proper operation of clustering with OpenStack services. First download RabbitMQ Server version 2.8.7:

```
wget -O /tmp/rabbitmq-server_2.8.7-1_all.deb http://www.rabbitmq.com/releases/rabbitmq-server/v2.8
```

Install RabbitMQ Server 2.8.7 dependencies:

```
apt-get install -y erlang-nox
```

Install RabbitMQ Server:

```
dpkg -i /tmp/rabbitmq-server_2.8.7-1_all.deb
```

Configure RabbitMQ Clustering. First, stop the rabbitmq-server service on all control nodes.

```
service rabbitmq-server stop
```

Clustering requires that the nodes have the same Erlang cookie. Copy the Erlang cookie from control01 to control02 and control03:

```
scp /var/lib/rabbitmq/.erlang.cookie localadmin@192.168.220.42:/var/lib/rabbitmq/.erlang.cookie
scp /var/lib/rabbitmq/.erlang.cookie localadmin@192.168.220.43:/var/lib/rabbitmq/.erlang.cookie
```

Note: The above command requires root login (disabled by default in Ubuntu). If you do not have root permissions, copy the Erlang cookie from /var/lib/rabbitmq/ to the /tmp directory of control02 and control03 and then to /var/lib/rabbitmq/. Also, make sure the file permissions match on all 3 nodes.

Now that all 3 control nodes have the same Erlang cookie, make sure that RabbitMQ will start:

```
service rabbitmq-server start
```

Note: If RabbitMQ does not successfully start, do not proceed with clustering.

Clustering can be configured using rabbitmqctl commands or by modifying the RabbitMQ configuration file. Our example uses the rabbitmqctl commands. You can see both approaches to configuring RabbitMQ clustering here. **Note:** Joining a cluster implicitly resets the node, thus removing all resources and data that were previously present on that node.

From control02:

```
rabbitmqctl stop_app
rabbitmqctl cluster rabbit@control01
rabbitmqctl start_app
```

Verify that control02 is now clustered with control01:

```
rabbitmqctl cluster_status

Cluster status of node rabbit@control02 ...
[{nodes,[{disc,[rabbit@control01]},{ram,[rabbit@control02]}]},
 {running_nodes,[rabbit@control01,rabbit@control02]}]
...done.
```

From control03:

```
rabbitmqctl stop_app
rabbitmqctl cluster rabbit@control02
rabbitmqctl start_app
```

Verify that control03 has joined the cluster:

```
rabbitmqctl cluster_status

Cluster status of node rabbit@control03 ...
[{nodes,[{disc,[rabbit@control01]},{ram,[rabbit@control03,rabbit@control02]}]},
 {running_nodes,[rabbit@control02,rabbit@control01,rabbit@control03]}]
...done.
```

Now that clustering is complete, secure RabbitMQ by removing the default (guest) user from only one of the nodes in the cluster:

```
rabbitmqctl delete_user guest
```

From only one of the nodes in the cluster, create a RabbitMQ user account that will be used by OpenStack services:

```
rabbitmqctl add_user openstack_rabbit_user openstack_rabbit_password
```

From only one of the nodes in the cluster, set the permissions for the new RabbitMQ user account:

```
rabbitmqctl set_permissions -p / openstack_rabbit_user ".*" ".*" ".*"
```

Verify the user settings:

```
rabbitmqctl list_users
rabbitmqctl list_user_permissions openstack_rabbit_user
```

Keystone Installation

Install Keystone on every control node:

```
apt-get install -y keystone python-keyring
```

Remove the sqllite db:

```
rm /var/lib/keystone/keystone.db
```

Create a MySQL database for Keystone. The database needs to be created on only 1 control node.

```
mysql
CREATE DATABASE keystone;
GRANT ALL ON keystone.* TO 'keystone_admin'@'%' IDENTIFIED BY 'keystone_db_pass';
GRANT ALL ON keystone.* TO 'keystone_admin'@'localhost' IDENTIFIED BY 'keystone_db_pass';
quit;
```

Note: From other controllers in the cluster, you can see that databases are replicated by Galera:

```
mysql -e "show databases;"
```

Edit the /etc/keystone/keystone.conf file on each controller. Change [CONTROLLER_MGT_IP] to the management IP address of the control node (i.e. control01: bind_host = 192.168.220.41):

```
[DEFAULT]
admin_token = keystone_admin_token
bind_host = [CONTROLLER_MGT_IP]
verbose = True

[sql]
connection = mysql://keystone_admin:keystone_db_pass@192.168.220.40/keystone
idle_timeout = 30

[ssl]
enable = False

[signing]
token_format = UUID
```

Create a credential file and load it so credentials are not required for every OpenStack client command. **Note:** This needs to be created on each node that you will run OpenStack commands from:

```
vi /root/openrc

export OS_TENANT_NAME=admin
export OS_USERNAME=admin
export OS_PASSWORD=keystone_admin
export OS_AUTH_URL="http://192.168.220.40:5000/v2.0/"
export OS_AUTH_STRATEGY=keystone
export SERVICE_TOKEN=keystone_admin_token
export SERVICE_ENDPOINT=http://192.168.220.40:35357/v2.0/

source /root/openrc
```

Verify that MySQL is listening on the VIP for the Keystone database. If you have any problems connecting to the VIP, try the real IP address of a control node:

```
mysql -h192.168.220.40 -ukeystone_admin -pkeystone_db_pass keystone
```

Restart Keystone:

```
service keystone restart
```

Synchronize the database on only one control node:

```
keystone-manage db_sync
```

Download the Keystone data script(Provided by Emilien Macchi):

```
wget https://raw.github.com/EmilienM/openstack-folsom-guide/master/scripts/keystone-data.sh
```

Edit the following fields in the script:

```
ADMIN_PASSWORD=${ADMIN_PASSWORD:-keystone_admin}
export SERVICE_TOKEN="keystone_admin_token"
export SERVICE_ENDPOINT="http://192.168.220.40:35357/v2.0/"
SERVICE_TENANT_NAME=${SERVICE_TENANT_NAME:-services}
```

Edit the file permissions

```
chmod +x keystone-data.sh
```

Run the script to populate the Keystone database with data (users, tenants, services). **Note:** If you see a long timeout and errors about "connection timeout", it may be related to your proxy setting. Remove the export of your http/https proxies and re-run the script. You will have to re-add your proxies for any other external downloads.

```
./keystone-data.sh
```

Note: You can ignore the following CLI message when running the scripts and any future Keystone commands:

WARNING: Bypassing authentication using a token & endpoint (authentication credentials are being ignored).

Download the Keystone endpoint script (Provided by Emilien Macchi):

```
wget https://raw.github.com/EmilienM/openstack-folsom-guide/master/scripts/keystone-endpoints.sh
```

Edit the following fields in the script:

```
# MySQL definitions
MYSQL_USER=keystone_admin
MYSQL_DATABASE=keystone
MYSQL_HOST=192.168.220.40
MYSQL_PASSWORD=keystone_db_pass

# Keystone definitions
KEYSTONE_REGION=RegionOne
SERVICE_TOKEN=keystone_admin_token
SERVICE_ENDPOINT="http://192.168.220.40:35357/v2.0"

# other definitions
MASTER="192.168.220.40"
SWIFT_MASTER="192.168.220.60"
```

Edit the file permissions

```
chmod +x keystone-endpoints.sh
```

Run the script to populate the Keystone database with service endpoints. Again, if you are using proxies then you will need remove them from your export before running this command:

```
./keystone-endpoints.sh
```

Test connectivity to Keystone by using a curl request :

```
apt-get install curl openssl -y

curl -d '{"auth": {"tenantName": "admin", "passwordCredentials":{"username": "admin", "password":
```

If the above command is successful, you will receive output that includes a token and a list of service endpoints. You may also want to verify the other service account credentials:

Glance

```
curl -s -d  "{\"auth\":{\"passwordCredentials\": {\"username\": \"glance\", \"password\": \"keysto
```

Nova

```
curl -s -d  "{\"auth\":{\"passwordCredentials\": {\"username\": \"nova\", \"password\": \"keystone
```

Swift

```
curl -s -d  "{\"auth\":{\"passwordCredentials\": {\"username\": \"swift\", \"password\": \"keyston
```

Quantum

```
curl -s -d  "{\"auth\":{\"passwordCredentials\": {\"username\": \"quantum\", \"password\": \"keyst
```

Cinder

```
curl -s -d  "{\"auth\":{\"passwordCredentials\": {\"username\": \"cinder\", \"password\": \"keysto
```

You can also use the Keystone client to verify the configuration:

```
keystone tenant-list
keystone user-list
keystone role-list
keystone service-list
keystone endpoint-list
```

Now that Keystone is operational, you may want to go back to the Verify the Swift Installation section to ensure that Swift is fully operational. If Swift is inoperable, you will be unable to add images to Glance in the next section.

Glance Installation

Install Glance API and Registry packages on all control nodes:

```
apt-get install -y glance-api glance-registry
```

Delete the glance.sqlite file created in the /var/lib/glance/ directory

```
rm /var/lib/glance/glance.sqlite
```

Create a MySQL database for Glance on only 1 control node:

```
mysql
CREATE DATABASE glance;
GRANT ALL ON glance.* TO 'glance'@'%' IDENTIFIED BY 'glance_pass';
GRANT ALL ON glance.* TO 'glance'@'localhost' IDENTIFIED BY 'glance_pass';
quit;
```

Edit the /etc/glance/glance-api.conf as follows. Replace [CONTROLLER_MGT_IP] with the controller management IP address (i.e. control01: bind_host = 192.168.220.41). Make changes on each control node.:

```
[DEFAULT]
verbose = True
default_store = swift
bind_host = [CONTROLLER_MGT_IP]
sql_connection=mysql://glance:glance_pass@192.168.220.40/glance
sql_idle_timeout = 30
registry_host = 192.168.220.40
swift_store_auth_address = http://192.168.220.40:5000/v2.0/
swift_store_user = services:swift
swift_store_key = keystone_admin
swift_store_container = glance
swift_store_create_container_on_put = True

[keystone_authtoken]
auth_host = 192.168.220.40
auth_port = 35357
auth_protocol = http
admin_tenant_name = services
admin_user = glance
admin_password = keystone_admin

[paste_deploy]
flavor=keystone+cachemanagement
```

Edit the /etc/glance/glance-registry.conf as follows. Replace [CONTROLLER_MGT_IP] with the controller management IP address (i.e. control01: bind_host = 192.168.220.41) Make changes on each control node.:

```
[DEFAULT]
verbose = True
bind_host = [CONTROLLER_MGT_IP]
sql_connection=mysql://glance:glance_pass@192.168.220.40/glance
sql_idle_timeout = 30

[keystone_authtoken]
auth_host = 192.168.220.40
auth_port = 35357
auth_protocol = http
admin_tenant_name = services
admin_user = glance
admin_password = keystone_admin

[paste_deploy]
flavor=keystone
```

Restart the glance-api and glance-registry services:

```
service glance-api restart; service glance-registry restart
```

The database tables are under version control and you use the following command on a new installation to prevent the Image service from breaking possible upgrades. This command is used on only one of the controllers:

```
glance-manage version_control 0
```

Synchronize the glance database on one control node (You may get a message about deprecation - you can ignore):

```
glance-manage db_sync
```

Restart the services again to take into account the new modifications:

```
service glance-registry restart; service glance-api restart
```

Download the Cirros 0.3.1 cloud image to a controller node and then upload it to Glance:

```
wget http://download.cirros-cloud.net/0.3.1/cirros-0.3.1-x86_64-disk.img

glance add name="cirros" is_public=true disk_format=qcow2 container_format=ovf < cirros-0.3.1-x86_
```

Verify that Glance is serving the image:

```
glance image-list
```

Optionally, you can add the Ubuntu Precise image to Glance:

```
wget http://cloud-images.ubuntu.com/precise/current/precise-server-cloudimg-amd64-disk1.img

glance add name="precise" is_public=true container_format=ovf disk_format=qcow2 < precise-server-c
```

Quantum Installation

Install the Quantum Server on all control nodes:

```
apt-get install -y quantum-server quantum-plugin-openvswitch
```

Create the Quantum database on only one control node:

```
mysql
CREATE DATABASE quantum;
GRANT ALL ON quantum.* TO 'quantum'@'%' IDENTIFIED BY 'quantum_pass';
GRANT ALL ON quantum.* TO 'quantum'@'localhost' IDENTIFIED BY 'quantum_pass';
quit;
```

Edit the /etc/quantum/quantum.conf file on all control nodes. Replace [CONTROLLER_MGT_IP] with the controller management IP address (i.e. control01: bind_host = 192.168.220.41):

```
[DEFAULT]
bind_host = [CONTROLLER_MGT_IP]
rabbit_userid=openstack_rabbit_user
rabbit_password=openstack_rabbit_password
rabbit_ha_queues=True
rabbit_hosts=control01:5672,control02:5672,control03:5672
verbose = True
```

```
log_file=/var/log/quantum/server.log

[keystone_authtoken]
auth_host = 192.168.220.40
auth_port = 35357
auth_protocol = http
admin_tenant_name = services
admin_user = quantum
admin_password = keystone_admin
signing_dir = /var/lib/quantum/keystone-signing
```

Edit the OVS plugin configuration file /etc/quantum/plugins/openvswitch/ovs_quantum_plugin.ini on all control nodes:

```
[DATABASE]
sql_connection=mysql://quantum:quantum_pass@192.168.220.40/quantum
sql_idle_timeout = 30

[OVS]
network_vlan_ranges = physnet1
bridge_mappings = physnet1:br-ex

[SECURITYGROUP]
firewall_driver = quantum.agent.linux.iptables_firewall.OVSHybridIptablesFirewallDriver
```

Restart the quantum server:

```
service quantum-server restart
```

Nova Installation

Start by installing the Nova software packages on all Control Nodes:

```
apt-get install -y nova-api nova-conductor nova-consoleauth nova-scheduler nova-novncproxy
```

Create the Nova database on only one control node:

```
mysql
CREATE DATABASE nova;
GRANT ALL ON nova.* TO 'nova'@'%' IDENTIFIED BY 'nova_pass';
GRANT ALL ON nova.* TO 'nova'@'localhost' IDENTIFIED BY 'nova_pass';
quit;
```

Modify the authtoken section in the /etc/nova/api-paste.ini file on each control node to include the following:

```
[filter:authtoken]
paste.filter_factory = keystoneclient.middleware.auth_token:filter_factory
auth_host = 192.168.220.40
auth_port = 35357
auth_protocol = http
admin_tenant_name = services
admin_user = nova
admin_password = keystone_admin
signing_dir = /tmp/keystone-signing-nova
# Workaround for https://bugs.launchpad.net/nova/+bug/1154809
auth_version = v2.0
```

Edit the /etc/nova/nova.conf file with the following. Replace [CONTROLLER_MGT_IP] with the controller node's management IP address (i.e. control01 = 192.168.220.41). Do this on each control node.:

```
[DEFAULT]
sql_idle_timeout=30
network_api_class=nova.network.quantumv2.api.API
quantum_url=http://192.168.220.40:9696
quantum_admin_auth_url=http://192.168.220.40:35357/v2.0
quantum_auth_strategy=keystone
quantum_admin_tenant_name=services
quantum_admin_username=quantum
quantum_admin_password=keystone_admin
firewall_driver=nova.virt.firewall.NoopFirewallDriver
service_quantum_metadata_proxy=true
quantum_metadata_proxy_shared_secret=quantum_proxy_secret
dhcpbridge_flagfile=/etc/nova/nova.conf
dhcpbridge=/usr/bin/nova-dhcpbridge
logdir=/var/log/nova
state_path=/var/lib/nova
lock_path=/var/lock/nova
iscsi_helper=tgtadm
libvirt_use_virtio_for_bridges=True
verbose=true
ec2_private_dns_show_ip=True
api_paste_config=/etc/nova/api-paste.ini
image_service=nova.image.glance.GlanceImageService
rpc_backend=nova.rpc.impl_kombu
rabbit_ha_queues=True
rabbit_hosts=control01:5672,control02:5672,control03:5672
glance_api_servers=192.168.220.40:9292
service_down_time=60
rabbit_port=5672
rabbit_virtual_host=/
sql_connection=mysql://nova:nova_pass@192.168.220.40/nova
memcached_servers=192.168.220.41:11211,192.168.220.42:11211,192.168.220.43:11211
rabbit_userid=openstack_rabbit_user
rabbit_password=openstack_rabbit_password
metadata_listen=[CONTROLLER_MGT_IP]
ec2_listen=[CONTROLLER_MGT_IP]
enabled_apis=ec2,osapi_compute,metadata
osapi_compute_listen=[CONTROLLER_MGT_IP]
volume_api_class=nova.volume.cinder.API
auth_strategy=keystone
rootwrap_config= /etc/nova/rootwrap.conf
novncproxy_port=6080
novncproxy_host=0.0.0.0
novncproxy_base_url=http://192.168.220.40:6080/vnc_auto.html
novncproxy_host=[CONTROLLER_MGT_IP]
```

Note: The nova.conf in our example enables verbose logging. When the environment is functional, you may want to consider changing verbose to false.

Synchronize the Nova database on only one control node (You may get a DEBUG message - You can ignore this):

```
nova-manage db sync
```

Due to bug 856764 Kombu must be patched to support channel_error detection of Rabbit queues. First, see if Kombu needs to be patched by grep'ing the file. You will receive no output if the file needs to be patched. You will receive self.channel_errors = self.connection.channel_errors if the file does **NOT** need patching:

```
grep self.channel_errors /usr/lib/python2.7/dist-packages/nova/openstack/common/rpc/impl_kombu.py
```

If the impl_kombu.py needs patching, download the patched file:

```
wget https://raw.github.com/CiscoSystems/puppet-coe/patch_support/files/impl_kombu.py.patch
```

Copy the patched impl_kombu.py to the /usr/lib/python2.7/dist-packages/nova/openstack/common/rpc/ directory:

```
cp impl_kombu.py /usr/lib/python2.7/dist-packages/nova/openstack/common/rpc/impl_kombu.py
```

Make sure the file is owned by root:root.

```
ls -l /usr/lib/python2.7/dist-packages/nova/openstack/common/rpc/impl_kombu.py
```

If impl_kombu.py is not owned by root, then change the file ownership:

```
chmod /usr/lib/python2.7/dist-packages/nova/openstack/common/rpc/impl_kombu.py
```

Restart nova-* services on all control nodes:

```
cd /etc/init.d/; for i in $( ls nova-* ); do sudo service $i restart; done
```

Check for the smiling faces on nova services to confirm your installation:

```
nova-manage service list
```

Also check that nova-api is running:

```
service nova-api status
```

Cinder Installation

Start by installing the Cinder software packages on all control nodes:

```
apt-get install -y cinder-api cinder-scheduler
```

Create the Cinder MySQL database on 1 control node:

```
mysql
CREATE DATABASE cinder;
GRANT ALL ON cinder.* TO 'cinder'@'%' IDENTIFIED BY 'cinder_pass';
GRANT ALL ON cinder.* TO 'cinder'@'localhost' IDENTIFIED BY 'cinder_pass';
quit;
```

Edit the /etc/cinder/api-paste.ini file on each control node.:

```
[filter:authtoken]
paste.filter_factory = keystoneclient.middleware.auth_token:filter_factory
service_protocol = http
service_host = 192.168.220.40
service_port = 5000
auth_host = 192.168.220.40
auth_port = 35357
auth_protocol = http
admin_tenant_name = services
admin_user = cinder
admin_password = keystone_admin
signing_dir = /var/lib/cinder
```

Edit the /etc/cinder/cinder.conf configuration file on each control node. **Note:** The default volume_group is being changed from cinder-volumes to nova-volumes. As mentioned in the Critical Reminders section, an LVM Volume Group named nova-volumes must exist on each Compute Node.

```
[DEFAULT]
sql_idle_timeout=30
rabbit_ha_queues=True
rabbit_hosts=control01:5672,control02:5672,control03:5672
rabbit_userid=openstack_rabbit_user
rabbit_password=openstack_rabbit_password
sql_connection = mysql://cinder:cinder_pass@192.168.220.40/cinder
osapi_volume_listen = [CONTROLLER_MGT_IP]
rootwrap_config = /etc/cinder/rootwrap.conf
api_paste_confg = /etc/cinder/api-paste.ini
iscsi_helper = tgtadm
volume_name_template = volume-%s
volume_group = nova-volumes
verbose = True
auth_strategy = keystone
state_path = /var/lib/cinder
lock_path = /var/lock/cinder
volumes_dir = /var/lib/cinder/volumes
```

Initialize the Cinder database on only one control node:

```
cinder-manage db sync
```

Restart Cinder services on all control nodes:

```
service cinder-api restart; service cinder-scheduler restart
```

Horizon Installation

Start by installing the Horizon software packages on all control nodes:

```
apt-get install -y memcached libapache2-mod-wsgi openstack-dashboard
```

Next, modify the /etc/openstack-dashboard/local_settings.py file. Replace all loopback interface definitions (i.e. 127.0.0.1) with the Controller Cluster VIP address (i.e. 192.168.220.40).

Change the memcached listening address in /etc/memcached.conf. Replace [CONTROLLER_MGT_IP] with the controller management IP address (i.e. control01 = 192.168.220.41):

```
-l [CONTROLLER_MGT_IP]
```

Reload Apache and memcached on each control node:

```
service apache2 restart; service memcached restart
```

Access Horizon by using the following URL in your web browser. Use **admin/keystone_admin** for your login credentials. If you have problems accessing Horizon by using the VIP (192.168.220.40), then try using a real IP address of a control node (i.e. control01 = 192.168.220.41):

```
http://192.168.220.40/horizon
```

Optionally, if you would like to remove the Ubuntu theme:

```
apt-get purge -y openstack-dashboard-ubuntu-theme
```

Compute Node Installation

Ensure you have completed the steps in the General Installation Steps for All Nodes section before proceeding. Follow these steps for compute01, compute02 and compute03 compute nodes.

Upgrade Client Libraries

Several client libraries must be upgraded to support RabbitMQ Mirrored Queues.

Update the required client libraries:

```
apt-get install -y python-pip
pip install kombu==2.4.7
pip install amqp==0.9.4
```

Check your version of anyjson:

```
pip freeze | grep anyjson
```

If anyjson is not 0.3.3, then install the correct version:

```
pip install anyjson==0.3.3
```

Quantum Installation

Install the Quantum software packages:

```
apt-get -y install quantum-plugin-openvswitch quantum-plugin-openvswitch-agent quantum-dhcp-agent
```

Check the status of the Open vSwitch services on each compute node:

```
service openvswitch-switch status
```

Start the Open vSwitch services on each compute node if they are not running:

```
service openvswitch-switch start
```

Compute Nodes require OVS bridges named "br-int" and "br-ex", and that "br-ex" is associated with the Public Network interface (eth1 in our example):

```
ovs-vsctl add-br br-int
ovs-vsctl add-br br-ex
ovs-vsctl add-port br-ex eth1
```

Edit the Quantum configuration file /etc/quantum/quantum.conf with the following. **Note:** Make sure the names in rabbit_hosts= resolve:

```
#Under the default section
[DEFAULT]
rabbit_userid=openstack_rabbit_user
rabbit_password=openstack_rabbit_password
rabbit_ha_queues=True
rabbit_hosts=control01:5672,control02:5672,control03:5672
verbose = True
```

```
log_file=/var/log/quantum/server.log

#Under the keystone_authtoken section
[keystone_authtoken]
auth_host = 192.168.220.40
auth_port = 35357
auth_protocol = http
admin_tenant_name = services
admin_user = quantum
admin_password = keystone_admin
signing_dir = /var/lib/quantum/keystone-signing
```

Edit the OVS plugin configuration file /etc/quantum/plugins/openvswitch/ovs_quantum_plugin.ini with:

```
#Under the database section
[DATABASE]
sql_idle_timeout = 30
```

Edit the following under the OVS section. **Note:** 223:225 signifies the VLAN ID range used for tenant VLANs. Modify this range based on your deployment needs. These VLANs should be trunked to eth1 of Compute Nodes and you must create a gateway address (i.e. 192.168.223.1 for VLAN 223) on your upstream Layer-3 device.

```
[OVS]
network_vlan_ranges = physnet1:223:225
bridge_mappings = physnet1:br-ex

# Using Quantum Security Groups instead of Nova Security Groups
[SECURITYGROUP]
firewall_driver = quantum.agent.linux.iptables_firewall.OVSHybridIptablesFirewallDriver
```

Update the /etc/quantum/dhcp_agent.ini:

```
#Under the default section
[DEFAULT]
# Required to run multiple Quantum DHCP agents
use_component_ext = True
# Add root_helper due to bug: https://bugs.launchpad.net/quantum/+bug/1182616
root_helper = sudo quantum-rootwrap /etc/quantum/rootwrap.conf
```

Update the /etc/quantum/metadata_agent.ini:

```
metadata_proxy_shared_secret = quantum_proxy_secret
```

Restart the Quantum services on each compute node:

```
service quantum-plugin-openvswitch-agent restart; service quantum-dhcp-agent restart; service quan
```

Nova Installation

Start by installing the Nova Compute software package on all Compute Nodes:

```
apt-get install -y nova-compute
```

Edit the /etc/nova/nova.conf file with the following. Replace [COMPUTE_MGT_IP] with the compute node's management IP address (i.e. compute01 = 192.168.220.51):

```
[DEFAULT]
```

```
force_config_drive=true
network_api_class=nova.network.quantumv2.api.API
quantum_url=http://192.168.220.40:9696
quantum_admin_auth_url=http://192.168.220.40:35357/v2.0
quantum_auth_strategy=keystone
quantum_admin_tenant_name=services
quantum_admin_username=quantum
quantum_admin_password=keystone_admin
firewall_driver=nova.virt.firewall.NoopFirewallDriver
libvirt_vif_driver=nova.virt.libvirt.vif.LibvirtHybridOVSBridgeDriver
service_quantum_metadata_proxy=true
quantum_metadata_proxy_shared_secret=quantum_proxy_secret
logdir=/var/log/nova
verbose=true
state_path=/var/lib/nova
lock_path=/var/lock/nova
iscsi_helper=tgtadm
libvirt_use_virtio_for_bridges=True
ec2_private_dns_show_ip=True
api_paste_config=/etc/nova/api-paste.ini
rabbit_ha_queues=True
rabbit_hosts=control01:5672,control02:5672,control03:5672
glance_api_servers=192.168.220.40:9292
sql_connection=mysql://nova:nova_pass@192.168.220.40/nova
memcached_servers=192.168.220.40:11211
rabbit_userid=openstack_rabbit_user
rabbit_password=openstack_rabbit_password
metadata_host=192.168.220.40
volume_api_class=nova.volume.cinder.API
auth_strategy=keystone
rootwrap_config= /etc/nova/rootwrap.conf
vncserver_proxyclient_address=[COMPUTE_MGT_IP]
novncproxy_base_url=http://192.168.220.40:6080/vnc_auto.html
live_migration_flag=VIR_MIGRATE_UNDEFINE_SOURCE,VIR_MIGRATE_PEER2PEER,VIR_MIGRATE_LIVE
```

Note: The nova.conf in our example enables verbose logging. When the environment is functional, you may want to consider changing verbose to false.

Verify that the /etc/nova/nova-compute.conf file looks like the following:

```
[DEFAULT]
libvirt_type=kvm
compute_driver=libvirt.LibvirtDriver
```

Restart the nova-compute service on each compute node:

```
service nova-compute restart
```

Create a credentials file so you can issue OpenStack client commands from the Compute Nodes:

```
vi /root/openrc

export OS_TENANT_NAME=admin
export OS_USERNAME=admin
export OS_PASSWORD=keystone_admin
export OS_AUTH_URL="http://192.168.220.40:5000/v2.0/"
export OS_AUTH_STRATEGY=keystone
export SERVICE_TOKEN=keystone_admin_token
export SERVICE_ENDPOINT=http://192.168.220.40:35357/v2.0/

source /root/openrc
```

Check for the smiling faces on nova services to confirm your installation:

```
nova-manage service list
```

Use the following steps if you would like to add support for migrating instances. The details of migration is outside the scope of this document. Use the official OpenStack documntation to understand the details of migration.

- Uncomment the following in /etc/libvirt/qemu.conf

```
cgroup_device_acl = [
"/dev/null", "/dev/full", "/dev/zero",
"/dev/random", "/dev/urandom",
"/dev/ptmx", "/dev/kvm", "/dev/kqemu",
"/dev/rtc", "/dev/hpet"
]
```

- Edit /etc/libvirt/libvirtd.conf file as follows:

```
listen_tls = 0
listen_tcp = 1
auth_tcp = "none"
```

- Modify libvirtd_opts variable in /etc/init/libvirt-bin.conf file :

```
env libvirtd_opts="-d -l"
```

- Edit /etc/default/libvirt-bin file :

```
libvirtd_opts="-d -l"
```

- Restart libvirt :

```
service libvirt-bin restart
```

Cinder Installation

Start by installing Cinder software packages on all Compute Nodes:

```
apt-get install -y cinder-volume
```

Edit the /etc/cinder/cinder.conf file with the following. Replace [COMPUTE_MGT_IP] with the compute node's management IP address (i.e. compute01 = 192.168.220.51):

```
[DEFAULT]
iscsi_ip_address=[COMPUTE_MGT_IP]
rabbit_ha_queues=True
rabbit_hosts=control01:5672,control02:5672,control03:5672
rabbit_userid=openstack_rabbit_user
rabbit_password=openstack_rabbit_password
sql_connection = mysql://cinder:cinder_pass@192.168.220.40/cinder
rootwrap_config = /etc/cinder/rootwrap.conf
api_paste_confg = /etc/cinder/api-paste.ini
iscsi_helper = tgtadm
volume_name_template = volume-%s
volume_group = nova-volumes
verbose = True
auth_strategy = keystone
```

```
state_path = /var/lib/cinder
lock_path = /var/lock/cinder
volumes_dir = /var/lib/cinder/volumes
```

Restart the Cinder services on all compute nodes:

```
service cinder-volume restart; service tgt restart
```

Configuring OpenStack Networking (Quantum) and Deploying the First VM

Run the following commands from either a Compute Node or Controller Node. If something has to be done on a specific node it will be called out. **Note:** If you have an issue with a Quantum command no being found, you may need to do the following:

```
apt-get install -y python-pip
pip install -U cliff
```

Create your first tenant network. In our example, we use the admin tenant. Create additional networks and associated subnets as needed. **Note:** The network is created with the "--shared" argument set so that the network is available to all tenants. If you only want this network available to the tenant for which you set your openrc file to then remove that argument:

```
keystone tenant-list
quantum net-create public223 --tenant_id admin --provider:network_type vlan --provider:physical_ne
```

Create your first tenant subnet and associate it to the network you created in the previous step. The example below uses .10-.250 for Instance IP addresses. Modify the allocation-pool and dns_nameservers based on your deployment needs.

```
quantum subnet-create --name 223-subnet --allocation-pool start=192.168.223.10,end=192.168.223.250
```

If you skipped the earlier step of downloading an image and uploading it to glance, do that now:

```
wget http://cloud-images.ubuntu.com/precise/current/precise-server-cloudimg-amd64-disk1.img

glance add name="precise" is_public=true container_format=ovf disk_format=qcow2 < precise-server-c
```

On a Compute Node create an SSH keypair and add the public key to Nova. **Note:** Leave the passphrase empty when creating the keypair. If you have an issue with the Nova commands not being found, you will need to to install the nova client support:

```
apt-get -y install python-novaclient

ssh-keygen

cd ~/.ssh/
nova keypair-add --pub_key id_rsa.pub <key_name>
```

Example:

```
nova keypair-add --pub_key id_rsa.pub net-key
```

Before booting the instance, check for the ID of the network we created earlier. Note: the <quantum_net_id> value will come from the output of the "quantum net-list" command:

```
quantum net-list
```

```
nova boot --image precise --flavor m1.small --key_name <key_name> --nic net-id=<quantum_net_id> <i
```

Example:

```
nova boot --image precise --flavor m1.small --key_name net-key --nic net-id=f9035744-72a9-42cf-bd4
```

Watch the status of the instance:

```
nova show <instance_name>
```

Example:

```
nova show vm1
```

The instance is booted completely when the OS-EXT-STS:vm_state is "active". Make note of the IP address of the VM. Alternatively, you can watch the complete log of the VM booting by running:

```
nova console-log --length=25 vm1
```

Note: If you see the instance log stuck in the following state, restart the quantum-dhcp-agent service on compute nodes and re-deploy your instance:

```
Starting network...
udhcpc (v1.20.1) started
Sending discover?
```

Edit the default Quantum Security Group to allow ingress traffic to Instances. <u</u>

Note: Security Group rules are associated to a specific tenant. We use the admin tenant in our example. The tenant-id in the command below should match the tenant-id in the previous quantum net-create command.

```
quantum security-group-rule-create default --direction ingress --ethertype IPv4 --protocol icmp --
```

```
quantum security-group-rule-create default --direction ingress --ethertype IPv4 --protocol tcp --p
```

This example allows all ICMP and SSH traffic:

```
quantum security-group-rule-create default --direction ingress --ethertype IPv4 --protocol icmp --
```

```
quantum security-group-rule-create default --direction ingress --ethertype IPv4 --protocol tcp --p
```

Note: If you receive the following message "*Multiple security_group matches found for name 'default', use an ID to be more specific.*", then replace the security-group name *default* with the security-group ID. The ID can be found by using the *quantum security-group-list* command.

You should now be able to ping the VM as well as SSH into it from the host that you used to create the key you associated with the VM during the boot phase.

Configuring OpenStack Networking (Quantum) DHCP Agent High-Availability

First, verify the status of your Quantum Agents. You should see :-) under alive and True under admin_state_up for all agents. Do not proceed with DHCP Agent fail-over testing if this is not the case:

```
quantum agent-list
+--------------------------------------+--------------------+-----------------------+-------+----
```

```
| id                                   | agent_type         | host                   | alive | adm
+--------------------------------------+--------------------+------------------------+-------+----
| 17538649-c80b-4c82-aedf-67ffca608a5d | DHCP agent         | compute03.dmz-pod2.lab | :-)   | Tru
| 4bc8dac3-ec4a-4369-b1b9-3c0111211d63 | DHCP agent         | compute02.dmz-pod2.lab | :-)   | Tru
| 4f574568-1342-4eea-94ae-96ce7b6af3f1 | Open vSwitch agent | compute01.dmz-pod2.lab | :-)   | Tru
| 53a44eae-025d-40a5-9505-10d33b3f9779 | DHCP agent         | compute01.dmz-pod2.lab | :-)   | Tru
| 7394b48c-5b1a-459f-95c0-1522c443fa88 | Open vSwitch agent | compute02.dmz-pod2.lab | :-)   | Tru
| a2f55922-b230-4e8f-8535-30ce59dbbb98 | Open vSwitch agent | compute03.dmz-pod2.lab | :-)   | Tru
+--------------------------------------+--------------------+------------------------+-------+----
```

Verify what DHCP Agent is servicing the Quantum public223 network:

```
quantum dhcp-agent-list-hosting-net public223
+--------------------------------------+------------------------+----------------+-------+
| id                                   | host                   | admin_state_up | alive |
+--------------------------------------+------------------------+----------------+-------+
| 4bc8dac3-ec4a-4369-b1b9-3c0111211d63 | compute02.dmz-pod2.lab | True           | :-)   |
+--------------------------------------+------------------------+----------------+-------+
```

Have another DHCP Agent service the public223 network. The DHCP Agent on Compute01 is being added to the public223 in our example. **Note:** The DHCP Agent ID can found in the output of the quantum agent-list command.

```
quantum dhcp-agent-network-add 53a44eae-025d-40a5-9505-10d33b3f9779 public223
Added network public223 to DHCP agent
```

Verify the DHCP Agent on Compute01 has been added to the public223 network:

```
root@control01:~# quantum dhcp-agent-list-hosting-net public223
+--------------------------------------+------------------------+----------------+-------+
| id                                   | host                   | admin_state_up | alive |
+--------------------------------------+------------------------+----------------+-------+
| 4bc8dac3-ec4a-4369-b1b9-3c0111211d63 | compute02.dmz-pod2.lab | True           | :-)   |
| 53a44eae-025d-40a5-9505-10d33b3f9779 | compute01.dmz-pod2.lab | True           | :-)   |
+--------------------------------------+------------------------+----------------+-------+
```

Verify that you can successfully boot a few Instances:

```
nova boot --image precise --flavor m1.small --key_name <key_name> --nic net-id=<quantum_net_id> <i
```

Example:

```
nova boot --image precise --flavor m1.small --key_name net-key --nic net-id=f9035744-72a9-42cf-bd4
```

Watch the status of the instance:

```
nova show <instance_name>
```

Example:

```
nova show vm1
```

The instance is booted completely when the OS-EXT-STS:vm_state is "active". Make note of the IP address of the VM. Alternatively, you can watch the complete log of the VM booting by running:

```
nova console-log --length=25 vm1
```

Note: Ensure that your Instance has successfully received an IP address from the nova console-log command. Test fail-over by disabling a DHCP Agent on one of the Compute Nodes. Our example disables the DHCP Agent on Compute02. **Note:** The DHCP Agent ID can found in the output of the quantum agent-list command.

```
quantum agent-update 4bc8dac3-ec4a-4369-b1b9-3c0111211d63 --admin_state_up=false
Updated agent: 4bc8dac3-ec4a-4369-b1b9-3c0111211d63
```

Verify the agent is disabled. You should see admin_state_up False associated to the DHCP Agent that you disabled:

```
root@control01:~# quantum agent-list
+--------------------------------------+--------------------+-----------------------+-------+----
| id                                   | agent_type         | host                  | alive | adm
+--------------------------------------+--------------------+-----------------------+-------+----
| 17538649-c80b-4c82-aedf-67ffca608a5d | DHCP agent         | compute03.dmz-pod2.lab | :-)   | Tru
| 4bc8dac3-ec4a-4369-b1b9-3c0111211d63 | DHCP agent         | compute02.dmz-pod2.lab | :-)   | Fal
| 4f574568-1342-4eea-94ae-96ce7b6af3f1 | Open vSwitch agent | compute01.dmz-pod2.lab | :-)   | Tru
| 53a44eae-025d-40a5-9505-10d33b3f9779 | DHCP agent         | compute01.dmz-pod2.lab | :-)   | Tru
| 7394b48c-5b1a-459f-95c0-1522c443fa88 | Open vSwitch agent | compute02.dmz-pod2.lab | :-)   | Tru
| a2f55922-b230-4e8f-8535-30ce59dbbb98 | Open vSwitch agent | compute03.dmz-pod2.lab | :-)   | Tru
+--------------------------------------+--------------------+-----------------------+-------+----
```

Verify that you can successfully boot a few Instances:

```
nova boot --image precise --flavor m1.small --key_name <key_name> --nic net-id=<quantum_net_id> <i
```

Example:

```
nova boot --image precise --flavor m1.small --key_name net-key --nic net-id=f9035744-72a9-42cf-bd4
```

Watch the status of the instance:

```
nova show <instance_name>
```

Example:

```
nova show vm1
```

The instance is booted completely when the OS-EXT-STS:vm_state is "active". Make note of the IP address of the VM. Alternatively, you can watch the complete log of the VM booting by running:

```
nova console-log --length=25 vm1
```

Note: Ensure that your Instance has successfully received an IP address from the nova console-log command.

Test fail-over by disabling a DHCP Agent on one of the Compute Nodes. Repeat the same process with other DHCP agents.

When fail-over testing is complete, enable all DHCP Agents and repeat the steps above for fail-back testing.

Support

Email: openstack-support@cisco.com

Credits

This work has been based on:

- OpenStack Grizzly Installation Guide for Ubuntu 12.04 (LTS) Documentation [1]

Authors

Daneyon Hansen

Shannon McFarland