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 Havana 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:

- Keystone (Identity Service)
- Glance (Image Service)
- Nova (Compute Service)
- Neutron (Network Service)
- Horizon (Network Service)
- Cinder (Block Storage Service)
- Swift (Object Storage Service)
- Heat (Orchestration 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 provides users step-by-step instructions for deploying an active/active, highly scalable OpenStack environment. 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 Neutron L3 and DHCP Agents Blueprint, as it states, allows multiple Neutron Layer-3 and DHCP Agents to be deployed for high-availability and scalability purposes. At this time, multiple DHCP Agents can service a Neutron network, however, only a single L3 Agent can service one Neutron 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. Neutron Provider Network Extensions are used to map physical data center networks to Neutron networks. In this deployment model, Neutron 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.

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 cinder-volumes (default value) 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.

Please read the release notes for the release you are installing. The release notes contain important information about limitations, features, and how to use snapshot repositories if you're installing an older release.

- h.0
- h.1
- h.2
- h.3

Operating System

The operating system used for this installation is Ubuntu 12.04 LTS (Precise).

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 Scheduler, Neutron Server, Neutron Plugin OVS, Neutron DHCP Agent, Glance API/Registry, Keystone, Cinder API, Cinder Scheduler, RabbitMQ Server, MySQL Server WSREP and Galera.

- Provides management functionality of the OpenStack environment.

3 Compute Nodes- Runs Nova Compute, Neutron OVS Agent, 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 Neutron 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 Neutron 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.

Reference Physical Network Configuration

Our reference architecture uses one Nexus 5000 per rack as a layer-2 top-of-rack switch. Each top of rack switch is connected to a Nexus 7000 switch that acts as a layer 3 gateway. It is recommended to use multiple layer-3 devices (i.e. Nexus 7000) and VRRP for redundancy.

Create VLANs on TOR L2 switch and L3 GW:

```
vlan 220
  name pod1_mgt
vlan 222
  name pod1_swift_storage
vlan 223
  name pod1-qtm--net1
vlan 224
  name pod1-qtm--net2
vlan 225
  name pod1-qtm--net3
```

Configure physical interfaces on the Nexus 5000 and 7000 switches. **Note:** This configuration applies to every interface within the path between the gateway interface and OpenStack nodes. Allowed VLANs can be further restricted based on OpenStack node type if needed:

```
interface <NAME/NUMBER>
  switchport mode trunk
  switchport trunk native vlan 220
  switchport trunk allowed vlan 220-230
  spanning-tree port type edge
```

Configure layer-3 gateway interfaces. VRRP configuration should be added to these interfaces if and additional physical layer-3 gateway is used. Notice that no layer-3 gateway interface has been created for the Swift storage network:

```
interface Vlan220
  ip address 192.168.220.1/24
  no shutdown

interface Vlan223
  ip address 192.168.223.1/24
  description ### Daneyon- Neutron Provider VLAN Deployment ###
  no shutdown

interface Vlan224
  ip address 192.168.224.1/24
  description ### Daneyon- Neutron Provider VLAN Deployment ###
  no shutdown

interface Vlan225
  ip address 192.168.225.1/24
  description ### Daneyon- Neutron Provider VLAN Deployment ###
  no shutdown
```

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 the reference deployment is physically connected to two Cisco Nexus switches that are configured for layer 2 only. eth0 of each node connects to Nexus switch 1 and eth1 of each node is connected to Nexus switch 2. Trunking is configured on each interface connecting to the eth0 and eth1 NICs of each node. Trunking is also configured between the switches and to the upstream physical routers.
- **Physical Network Routers:** Each Nexus switch is connected to a pair of upstream ASR 9000 routers. The routers provide layer 3 functionality, including first-hop redundancy (using VRRP) for all OpenStack networks. Keep in mind that the HA architecture uses Neutron Provider Networking Extensions, so tenant networks will also use ASR 9000 VRRP addresses as gateways.

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 Neutron 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 Neutron 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 cinder-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.

Havana Packages

The Cisco Havana repository uses packages built from the upstream git repositories and should be used for testing purposes only. The repository 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 Cisco Havanarepository by adding the following to /etc/apt/sources.list.d/cisco-openstack-mirror-havana.list:

```
deb http://openstack-repo.cisco.com/openstack/cisco havana main
deb-src http://openstack-repo.cisco.com/openstack/cisco havana main
deb http://openstack-repo.cisco.com/openstack/cisco_supplemental havana main
deb-src http://openstack-repo.cisco.com/openstack/cisco_supplemental havana main
```

If you receive the following error when running apt-get update:

```
GPG error: ftp://ftpeng.cisco.com folsom InRelease:
The following signatures couldn't be verified because the public key is not available:
NO_PUBKEY E8CC67053ED3B199
```

Then add the apt key to remove the error:

```
gpg --keyserver hkp://pgpkeys.mit.edu --recv-keys E8CC67053ED3B199
gpg --armor --export E8CC67053ED3B199 | apt-key add -
```

Note: If you have issues using the pgpkeys.mit.edu site you can use keyserver.ubuntu.com instead:

```
gpg --keyserver hkp://keyserver.ubuntu.com:80 --recv-keys E8CC67053ED3B199
```

Update your system:

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

Reboot your system:

```
reboot now
```

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. Additionally, the examples below use two separate physical interfaces on Control and Compute Nodes. For deployments requiring a single physical NIC on Control/Compute Nodes, reference the Single Physical NIC Section for configuration examples and additional deployment details.

```
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

# 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
```

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

# 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
```

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

# 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 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
  stats  socket /var/lib/haproxy/stats
  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 cinder_api_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 dashboard_cluster
  bind 192.168.220.40:80
  balance  source
  capture  cookie vgnvisitor= len 32
  cookie  SERVERID insert indirect nocache
  mode  http
  option  forwardfor
  option  httpchk
  option  httpclose
  rspidel  ^Set-cookie:\ IP=
  server control01 192.168.220.41:80 cookie control01 check inter 2000 rise 2 fall 5
  server control02 192.168.220.42:80 cookie control02 check inter 2000 rise 2 fall 5
  server control03 192.168.220.43:80 cookie control03 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 nova_ec2_api_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_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_metadata_api_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_osapi_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 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 neutron_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 heat_api_cluster
  bind 192.168.220.40:8004
  balance  source
  option  tcpka
  option  httpchk
  option  tcplog
  server control01 192.168.220.41:8004  check inter 2000 rise 2 fall 5
  server control02 192.168.220.42:8004  check inter 2000 rise 2 fall 5
  server control03 192.168.220.43:8004  check inter 2000 rise 2 fall 5

listen heat_cf_api_cluster
  bind 192.168.220.40:8000
  balance  source
  option  tcpka
  option  httpchk
  option  tcplog
  server control01 192.168.220.41:8000  check inter 2000 rise 2 fall 5
  server control02 192.168.220.42:8000  check inter 2000 rise 2 fall 5
  server control03 192.168.220.43:8000  check inter 2000 rise 2 fall 5

listen rabbit_cluster
  bind 192.168.220.40:5672
  balance  source
  mode  tcp
  option  tcpka
  option  tcplog
  server control01 192.168.220.41:5672  check inter 2000 rise 2 fall 5
  server control02 192.168.220.42:5672  check inter 2000 rise 2 fall 5
  server control03 192.168.220.43:5672  check inter 2000 rise 2 fall 5

listen swift_memcached_cluster
  bind 192.168.220.60:11211
  balance  source
  option  tcpka
  option  tcplog
  server swiftproxy01 192.168.220.61:11211  check inter 2000 rise 2 fall 5
  server swiftproxy02 192.168.220.62:11211  check inter 2000 rise 2 fall 5

listen swift_proxy_cluster
  bind 192.168.220.60:8080
  balance  source
  option  tcpka
```

```
  option  tcplog
  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
  stats  socket /var/lib/haproxy/stats
  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 cinder_api_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 dashboard_cluster
  bind 192.168.220.40:80
  balance  source
  capture  cookie vgnvisitor= len 32
  cookie  SERVERID insert indirect nocache
  mode  http
  option  forwardfor
  option  httpchk
  option  httpclose
  rspidel  ^Set-cookie:\ IP=
  server control01 192.168.220.41:80 cookie control01 check inter 2000 rise 2 fall 5
  server control02 192.168.220.42:80 cookie control02 check inter 2000 rise 2 fall 5
  server control03 192.168.220.43:80 cookie control03 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 nova_ec2_api_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_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_metadata_api_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_osapi_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 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 neutron_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 heat_api_cluster
  bind 192.168.220.40:8004
  balance  source
  option  tcpka
  option  httpchk
  option  tcplog
  server control01 192.168.220.41:8004  check inter 2000 rise 2 fall 5
  server control02 192.168.220.42:8004  check inter 2000 rise 2 fall 5
  server control03 192.168.220.43:8004  check inter 2000 rise 2 fall 5

listen heat_cf_api_cluster
  bind 192.168.220.40:8000
  balance  source
  option  tcpka
  option  httpchk
  option  tcplog
  server control01 192.168.220.41:8000  check inter 2000 rise 2 fall 5
  server control02 192.168.220.42:8000  check inter 2000 rise 2 fall 5
  server control03 192.168.220.43:8000  check inter 2000 rise 2 fall 5

listen rabbit_cluster
  bind 192.168.220.40:5672
  balance  source
  mode  tcp
  option  tcpka
  option  tcplog
  server control01 192.168.220.41:5672  check inter 2000 rise 2 fall 5
  server control02 192.168.220.42:5672  check inter 2000 rise 2 fall 5
  server control03 192.168.220.43:5672  check inter 2000 rise 2 fall 5

listen swift_memcached_cluster
  bind 192.168.220.60:11211
  balance  source
  option  tcpka
  option  tcplog
```

```
  server swiftproxy01 192.168.220.61:11211  check inter 2000 rise 2 fall 5
  server swiftproxy02 192.168.220.62:11211  check inter 2000 rise 2 fall 5

listen swift_proxy_cluster
  bind 192.168.220.60:8080
  balance  source
  option  tcpka
  option  tcplog
  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 required packages:

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

At the time of this writing, Swift services require dnspython >= 1.10.0. Therefore, verify the installed version of dnspython:

```
dpkg -l | grep dnspython
ii  python-dnspython                1.11.0-1ubuntu1                    DNS toolkit for Python
```

If dnspython is not >= 1.10.0, then install pip and the latest stable dnspython package:

```
apt-get install python-pip
pip install dnspython==1.11.1
```

Create the Swift configuration directory:

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

Create the Swift recon cache directory and set its permission:

```
mkdir -p /var/swift/recon
chown -R swift:swift /var/swift/recon
```

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 -s /dev/sdb mklabel gpt
mkfs.xfs -f -i size=1024 /dev/sdb
echo "/dev/sdb /srv/node/sdb xfs noatime,nodiratime,nobarrier,logbufs=8 0 0" >> /etc/fstab
mkdir -p /srv/node/sdb
mount /srv/node/sdb
```

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]
```

At this point, the ring files will not be present on the storage nodes. This will cause the Swift *-replicator and container-sync services to not start properly. After you create the ring files on the first proxy node (in the next section) and distribute them to the storage nodes, restarting the object-expirer, *-replicator and container-sync services will allow the remaining Swift storage node services to start properly. Start the following Swift services:

```
swift-init object-updater start
swift-init object-server start
swift-init object-auditor start
swift-init container-server start
swift-init container-updater start
swift-init container-auditor start
swift-init account-server start
swift-init account-auditor start
swift-init account-reaper 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. **Note:** Change the IP addresses of auth_host and memcache_servers if you are using your own IP addressing scheme.

```
[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, create ring entries for every storage device on each storage node. This example prepares

the account, container and object rings for storage nodes swift01 (192.168.222.71), swift02 (192.168.222.72) and swift03 (192.168.222.73) with a separate zone for each storage node. The mount points on each storage node are /srv/node/sdb thru /srv/node/sdf and the path in rsyncd.conf is /srv/node/. The Swift storage device would be sdb thru sdf and the commands would look like:

```
swift-ring-builder account.builder add z1-192.168.222.71:6002/sdb 100
swift-ring-builder container.builder add z1-192.168.222.71:6001/sdb 100
swift-ring-builder object.builder add z1-192.168.222.71:6000/sdb 100
swift-ring-builder account.builder add z1-192.168.222.71:6002/sdc 100
swift-ring-builder container.builder add z1-192.168.222.71:6001/sdc 100
swift-ring-builder object.builder add z1-192.168.222.71:6000/sdc 100
swift-ring-builder account.builder add z1-192.168.222.71:6002/sdd 100
swift-ring-builder container.builder add z1-192.168.222.71:6001/sdd 100
swift-ring-builder object.builder add z1-192.168.222.71:6000/sdd 100
swift-ring-builder account.builder add z1-192.168.222.71:6002/sde 100
swift-ring-builder container.builder add z1-192.168.222.71:6001/sde 100
swift-ring-builder object.builder add z1-192.168.222.71:6000/sde 100
swift-ring-builder account.builder add z1-192.168.222.71:6002/sdf 100
swift-ring-builder container.builder add z1-192.168.222.71:6001/sdf 100
swift-ring-builder object.builder add z1-192.168.222.71:6000/sdf 100
swift-ring-builder account.builder add z2-192.168.222.72:6002/sdb 100
swift-ring-builder container.builder add z2-192.168.222.72:6001/sdb 100
swift-ring-builder object.builder add z2-192.168.222.72:6000/sdb 100
swift-ring-builder account.builder add z2-192.168.222.72:6002/sdc 100
swift-ring-builder container.builder add z2-192.168.222.72:6001/sdc 100
swift-ring-builder object.builder add z2-192.168.222.72:6000/sdc 100
swift-ring-builder account.builder add z2-192.168.222.72:6002/sdd 100
swift-ring-builder container.builder add z2-192.168.222.72:6001/sdd 100
swift-ring-builder object.builder add z2-192.168.222.72:6000/sdd 100
swift-ring-builder account.builder add z2-192.168.222.72:6002/sde 100
swift-ring-builder container.builder add z2-192.168.222.72:6001/sde 100
swift-ring-builder object.builder add z2-192.168.222.72:6000/sde 100
swift-ring-builder account.builder add z2-192.168.222.72:6002/sdf 100
swift-ring-builder container.builder add z2-192.168.222.72:6001/sdf 100
swift-ring-builder object.builder add z2-192.168.222.72:6000/sdf 100
swift-ring-builder account.builder add z3-192.168.222.73:6002/sdb 100
swift-ring-builder container.builder add z3-192.168.222.73:6001/sdb 100
swift-ring-builder object.builder add z3-192.168.222.73:6000/sdb 100
swift-ring-builder account.builder add z3-192.168.222.73:6002/sdc 100
swift-ring-builder container.builder add z3-192.168.222.73:6001/sdc 100
swift-ring-builder object.builder add z3-192.168.222.73:6000/sdc 100
swift-ring-builder account.builder add z3-192.168.222.73:6002/sdd 100
swift-ring-builder container.builder add z3-192.168.222.73:6001/sdd 100
swift-ring-builder object.builder add z3-192.168.222.73:6000/sdd 100
swift-ring-builder account.builder add z3-192.168.222.73:6002/sde 100
swift-ring-builder container.builder add z3-192.168.222.73:6001/sde 100
swift-ring-builder object.builder add z3-192.168.222.73:6000/sde 100
swift-ring-builder account.builder add z3-192.168.222.73:6002/sdf 100
swift-ring-builder container.builder add z3-192.168.222.73:6001/sdf 100
swift-ring-builder object.builder add z3-192.168.222.73:6000/sdf 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:/etc/swift# swift-ring-builder /etc/swift/account.builder
/etc/swift/account.builder, build version 15
262144 partitions, 3.000000 replicas, 1 regions, 3 zones, 15 devices, 100.00 balance
The minimum number of hours before a partition can be reassigned is 1
Devices:    id  region  zone      ip address  port  replication ip  replication port      name wei
             0       1     1  192.168.222.71  6002  192.168.222.71              6002       sdb 100
             1       1     1  192.168.222.71  6002  192.168.222.71              6002       sdc 100
             2       1     1  192.168.222.71  6002  192.168.222.71              6002       sdd 100
             3       1     1  192.168.222.71  6002  192.168.222.71              6002       sde 100
             4       1     1  192.168.222.71  6002  192.168.222.71              6002       sdf 100
             5       1     2  192.168.222.72  6002  192.168.222.72              6002       sdb 100
             6       1     2  192.168.222.72  6002  192.168.222.72              6002       sdc 100
             7       1     2  192.168.222.72  6002  192.168.222.72              6002       sdd 100
             8       1     2  192.168.222.72  6002  192.168.222.72              6002       sde 100
             9       1     2  192.168.222.72  6002  192.168.222.72              6002       sdf 100
            10       1     3  192.168.222.73  6002  192.168.222.73              6002       sdb 100
            11       1     3  192.168.222.73  6002  192.168.222.73              6002       sdc 100
            12       1     3  192.168.222.73  6002  192.168.222.73              6002       sdd 100
            13       1     3  192.168.222.73  6002  192.168.222.73              6002       sde 100
            14       1     3  192.168.222.73  6002  192.168.222.73              6002       sdf 100

root@swiftproxy01:/etc/swift# swift-ring-builder /etc/swift/container.builder
/etc/swift/container.builder, build version 15
262144 partitions, 3.000000 replicas, 1 regions, 3 zones, 15 devices, 100.00 balance
The minimum number of hours before a partition can be reassigned is 1
Devices:    id  region  zone      ip address  port  replication ip  replication port      name wei
             0       1     1  192.168.222.71  6001  192.168.222.71              6001       sdb 100
             1       1     1  192.168.222.71  6001  192.168.222.71              6001       sdc 100
             2       1     1  192.168.222.71  6001  192.168.222.71              6001       sdd 100
             3       1     1  192.168.222.71  6001  192.168.222.71              6001       sde 100
             4       1     1  192.168.222.71  6001  192.168.222.71              6001       sdf 100
             5       1     2  192.168.222.72  6001  192.168.222.72              6001       sdb 100
             6       1     2  192.168.222.72  6001  192.168.222.72              6001       sdc 100
             7       1     2  192.168.222.72  6001  192.168.222.72              6001       sdd 100
             8       1     2  192.168.222.72  6001  192.168.222.72              6001       sde 100
             9       1     2  192.168.222.72  6001  192.168.222.72              6001       sdf 100
            10       1     3  192.168.222.73  6001  192.168.222.73              6001       sdb 100
            11       1     3  192.168.222.73  6001  192.168.222.73              6001       sdc 100
            12       1     3  192.168.222.73  6001  192.168.222.73              6001       sdd 100
            13       1     3  192.168.222.73  6001  192.168.222.73              6001       sde 100
            14       1     3  192.168.222.73  6001  192.168.222.73              6001       sdf 100

root@swiftproxy01:/etc/swift# swift-ring-builder /etc/swift/object.builder
/etc/swift/object.builder, build version 15
262144 partitions, 3.000000 replicas, 1 regions, 3 zones, 15 devices, 100.00 balance
The minimum number of hours before a partition can be reassigned is 1
Devices:    id  region  zone      ip address  port  replication ip  replication port      name wei
             0       1     1  192.168.222.71  6000  192.168.222.71              6000       sdb 100
             1       1     1  192.168.222.71  6000  192.168.222.71              6000       sdc 100
             2       1     1  192.168.222.71  6000  192.168.222.71              6000       sdd 100
             3       1     1  192.168.222.71  6000  192.168.222.71              6000       sde 100
             4       1     1  192.168.222.71  6000  192.168.222.71              6000       sdf 100
             5       1     2  192.168.222.72  6000  192.168.222.72              6000       sdb 100
             6       1     2  192.168.222.72  6000  192.168.222.72              6000       sdc 100
             7       1     2  192.168.222.72  6000  192.168.222.72              6000       sdd 100
             8       1     2  192.168.222.72  6000  192.168.222.72              6000       sde 100
             9       1     2  192.168.222.72  6000  192.168.222.72              6000       sdf 100
            10       1     3  192.168.222.73  6000  192.168.222.73              6000       sdb 100
```

```
  11       1     3  192.168.222.73  6000  192.168.222.73              6000       sdc 100
  12       1     3  192.168.222.73  6000  192.168.222.73              6000       sdd 100
  13       1     3  192.168.222.73  6000  192.168.222.73              6000       sde 100
  14       1     3  192.168.222.73  6000  192.168.222.73              6000       sdf 100
```

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
```

Restart the Proxy service on both Proxy nodes:

```
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 and verify the status of all Swift services:

```
swift-init status all
swift-init restart all
swift-init status all
```

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 the MySQL and Galera packages:

```
apt-get install -y mysql-server-wsrep galera python-mysqldb
```

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_user:wsrep_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_user:wsrep_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_user:wsrep_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. If the second, third nodes will not join the cluster, you may need to set wsrep_sst_receive_address to the same IP as the bind address.

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="monitor_user"
MYSQL_PASSWORD="monitor_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('%','monitor_user',PASSWORD('monitor_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 'monitor_user'@'[CONTROLLER_MGT_IP]' IDENTIFIED BY 'monitor_password
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 -umonitor_user -pmonitor_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=monitor_user --password=monitor_password -e "
```

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. RabbitMQ Server version 2.8.7 is the most widely tested and is included in the Cisco package repository. Install RabbitMQ server:

```
apt-get install -y rabbitmq-server
```

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. **Note:** Our example uses the root account to copy the files. Use the command *passwd root* on the control nodes if a root password has was not previously created.

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

Make sure the file permissions for the Erlang cookie match on all 3 nodes.

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

```
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
```

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]

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

[token]
provider = keystone.token.providers.uuid.Provider
```

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:

```
wget https://raw.github.com/danehans/openstack-guide/havana/scripts/keystone-data.sh
```

If you are not following the IP addressing scheme within the guide, then change 192.168.220.40 to the VIP of your controller cluster within keystone-data.sh.

```
vi keystone-data.sh
```

Edit the file permissions to make the script executable:

```
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:

```
wget https://raw.github.com/danehans/openstack-guide/havana/scripts/keystone-endpoints.sh
```

If you are not following the IP addressing scheme within the guide, then change 192.168.220.40 to the VIP of your controller cluster and 192.168.220.60 to the VIP of your Swift proxy cluster within keystone-endpoints.sh.

```
vi keystone-endpoints.sh
```

Edit the file permissions to make the script executable:

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

Run the script to populate the Keystone database with service endpoints. Again, if you are using proxies then remove them from your export before running this command. If you receive a user authentication error when running the script, you may need to run the keystone-endpoints.sh script with the -E [AUTH_URL] and -T [AUTH_TOKEN] flags (i.e. -E "http://192.168.220.40:35357/v2.0" -T keystone_admin_token).

```
./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
```

Neutron

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

Cinder

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

Heat

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

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 Glance_Installation 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]
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-cache.conf as follows:

```
[DEFAULT]
registry_host = 192.168.220.40
auth_url = http://192.168.220.40:5000/v2.0/
admin_tenant_name = services
admin_user = glance
admin_password = keystone_admin
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
```

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]
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
```

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 image-create --name cirros --is-public=true --disk-format=qcow2 --container-format=ovf < ci
```

Verify that Glance is serving the image:

```
glance image-list
```

Neutron Installation

Install the Neutron Server, DHCP Agent and OVS Plugin/Agent on all control nodes:

```
apt-get install -y neutron-server neutron-plugin-openvswitch neutron-plugin-openvswitch-agent oper
```

Create the Neutron database on only one control node:

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

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

```
service openvswitch-switch status
```

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

```
service openvswitch-switch start
```

Control 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 /etc/neutron/neutron.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

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

[database]
sql_connection=mysql://neutron:neutron_pass@192.168.220.40/neutron
```

Edit the OVS plugin configuration file /etc/neutron/plugins/openvswitch/ovs_neutron_plugin.ini on all control nodes. **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 Control Nodes and you must create a gateway address (i.e. 192.168.223.1 for VLAN 223) on your upstream Layer-3 device.

```
[ovs]
tenant_network_type = vlan
network_vlan_ranges = physnet1:223:225
bridge_mappings = physnet1:br-ex

[securitygroup]
firewall_driver = neutron.agent.linux.iptables_firewall.OVSHybridIptablesFirewallDriver
```

Make sure the OVS interface_driver is set in /etc/neutron/dhcp_agent.ini:

```
interface_driver = neutron.agent.linux.interface.OVSInterfaceDriver
```

Restart Neutron services:

```
service neutron-server restart; service neutron-dhcp-agent restart; service neutron-plugin-openvsw
```

Nova Installation

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

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

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
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.neutronv2.api.API
neutron_url=http://192.168.220.40:9696
neutron_admin_auth_url=http://192.168.220.40:35357/v2.0
neutron_auth_strategy=keystone
neutron_admin_tenant_name=services
neutron_admin_username=neutron
neutron_admin_password=keystone_admin
firewall_driver=nova.virt.firewall.NoopFirewallDriver
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
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=[CONTROLLER01_MGT_IP]:11211,[CONTROLLER02_MGT_IP]:11211,[CONTROLLER03_MGT_IP]:11
rabbit_userid=openstack_rabbit_user
rabbit_password=openstack_rabbit_password
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]
```

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

```
nova-manage db sync
```

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
```

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 documentation 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 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/keystone-signing
```

Edit the /etc/cinder/cinder.conf configuration file on each control node. **Note:** As mentioned in the Critical Reminders section, an LVM Volume Group named cinder-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 = cinder-volumes
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
```

Heat Installation

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

```
apt-get install -y heat-api heat-api-cfn heat-api-cloudwatch heat-engine
```

Create the Cinder MySQL database on 1 control node:

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

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

```
[filter:authtoken]
paste.filter_factory = heat.common.auth_token:filter_factory
service_host = 192.168.220.40
service_port = 5000
service_protocol = http
auth_host = 192.168.220.40
auth_port = 35357
auth_protocol = http
```

```
auth_uri = http://192.168.220.40:5000/v2.0/
admin_token = keystone_admin_token
```

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

```
[DEFAULT]
heat_metadata_server_url=http://192.168.220.40:8000
heat_waitcondition_server_url=http://192.168.220.40:8000/v1/waitcondition
heat_watch_server_url=http://192.168.220.40:8003
sql_idle_timeout=30
sql_connection = mysql://heat:heat_pass@192.168.220.40/heat
rabbit_ha_queues=True
rabbit_hosts=control01:5672,control02:5672,control03:5672
rabbit_userid=openstack_rabbit_user
rabbit_password=openstack_rabbit_password
#Ubuntu packages do not correctly set up logging.
log_dir=/var/log/heat
[ec2authtoken]
http://192.168.220.40:5000/v2.0/ec2tokens
[heat_api]
bind_host=[CONTROLLER_MGT_IP]
[heat_api_cfn]
bind_host=[CONTROLLER_MGT_IP]
[heat_api_cloudwatch]
bind_host=[CONTROLLER_MGT_IP]
```

Initialize the Heat database on only one control node:

```
heat-manage db_sync
```

Restart Heat services on all control nodes:

```
service heat-api restart; service heat-api-cfn restart; service heat-api-cloudwatch restart; servi
```

After you complete the HA deployment, use the official OpenStack guide to create Heat stacks.

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.

Neutron Installation

Install the Neutron software packages:

```
apt-get -y install neutron-plugin-openvswitch neutron-plugin-openvswitch-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 Neutron configuration file /etc/neutron/neutron.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

#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 = neutron
admin_password = keystone_admin
signing_dir = /var/lib/neutron/keystone-signing
```

Edit the OVS plugin configuration file /etc/neutron/plugins/openvswitch/ovs_neutron_plugin.ini to include the following. **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]
tenant_network_type = vlan
network_vlan_ranges = physnet1:223:225
bridge_mappings = physnet1:br-ex

# Using Neutron Security Groups instead of Nova Security Groups
[securitygroup]
firewall_driver = neutron.agent.linux.iptables_firewall.OVSHybridIptablesFirewallDriver
```

Restart the Neutron OVS Plugin Agent service on each compute node:

```
service neutron-plugin-openvswitch-agent restart
```

Nova Installation

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

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

Check to make sure /dev/kvm exists:

```
kvm-ok
```

If not, add it and restart nova-compute:

```
modprobe kvm_intel
service nova-compute restart
```

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.neutronv2.api.API
neutron_url=http://192.168.220.40:9696
neutron_admin_auth_url=http://192.168.220.40:35357/v2.0
neutron_auth_strategy=keystone
neutron_admin_tenant_name=services
neutron_admin_username=neutron
neutron_admin_password=keystone_admin
firewall_driver=nova.virt.firewall.NoopFirewallDriver
libvirt_vif_driver=nova.virt.libvirt.vif.LibvirtHybridOVSBridgeDriver
logdir=/var/log/nova
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
volume_api_class=nova.volume.cinder.API
auth_strategy=keystone
```

```
rootwrap_config= /etc/nova/rootwrap.conf
live_migration_flag=VIR_MIGRATE_UNDEFINE_SOURCE,VIR_MIGRATE_PEER2PEER,VIR_MIGRATE_LIVE
vnc_enabled=true
vncserver_listen=[COMPUTE_MGT_IP]
vncserver_proxyclient_address=[COMPUTE_MGT_IP]
novncproxy_base_url=http://192.168.220.40:6080/vnc_auto.html
```

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

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.

- 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 = cinder-volumes
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 (Neutron) 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.

Create your first tenant network. In our example, we use the admin tenant. Create additional networks as needed. **Note:** The *--tenant_id* flag is not specified in the following commands because we previously sourced our credential file.

```
neutron net-create public223 --provider:network_type vlan --provider:physical_network physnet1 --p
```

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. Create additional networks as needed.

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

Edit the default Neutron Security Group to allow ingress traffic to Instances. Our example allows all ingress ICMP and SSH traffic. **Note:** Security Group rules are associated to a specific tenant.

```
neutron security-group-rule-create default --direction ingress --ethertype IPv4 --protocol icmp --
neutron 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 *neutron security-group-list* command.

If you skipped the earlier step of downloading an image and uploading it to glance, do that now. **Note:** The HA architecture uses Config Drive instead of Nova Metadata, so make sure you use an image that supports Cloud Init.

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

glance image-create --name cirros --is-public=true --disk-format=qcow2 --container-format=ovf < ci
```

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

```
neutron net-list

nova boot --image cirros --flavor m1.tiny --nic net-id=<neutron_net_id> <instance_name>
```

Example:

```
nova boot --image cirros --flavor m1.tiny --nic net-id=f9035744-72a9-42cf-bd46-73d54c0cea06 vm1
```

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 neutron-dhcp-agent service on compute nodes and re-deploy your instance:

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

From a host with connectivity to your Neutron public subnet (192.168.223.0/24 in our example), you should now be able to ping and SSH to the VM. **Note:** The cirros image uses cirros/cubswin:) as the default SSH login credentials:

```
# ssh cirros@192.168.223.17
The authenticity of host '192.168.223.17 (192.168.223.17)' can't be established.
RSA key fingerprint is 42:49:28:17:67:ed:b2:63:f8:66:a6:3a:e8:0a:1a:01.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.223.17' (RSA) to the list of known hosts.
cirros@192.168.223.17's password:
$
```

Configuring SSH Key Injection for Accessing Instances

Config Drive and Cloud Init provide the ability to inject user and system information into an Instance during the boot-up process. SSH keys can be injected into Instances to provide password-less SSH access. A high-level breakdown of the process is as follows:

1. Create an SSH key-pair that will be used for injection.
2. Add the public key from the key-pair to Nova.
3. Reference the Nova key-pair name within the Nova boot command or Horizon GUI.
4. The public key will be injected into the instance by Config Drive and cloud-init during the boot-up.
5. The SSH client references the associated private key during the connection request.

From a host with connectivity to the Neutron public subnet (192.168.223.0/24 in our example), generate an SSH key-pair. **Note:** Leave the passphrase empty when creating the key-pair.

```
# ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_rsa.
Your public key has been saved in /root/.ssh/id_rsa.pub.
The key fingerprint is:
fd:d1:d4:6b:ab:df:03:9b:9d:64:6f:2b:2b:47:ae:d3 root@ubuntu-mgmt
The key's randomart image is:
+--[ RSA 2048]----+
|                 |
|                . |
|               . .|
|          .   o  .|
|         S . . .o |
|            . +.o.|
|             = B.o|
|            o E.++|
|            .=o+++|
+-----------------+
```

Add the SSH **public (.pub)** key to Nova. If the host being used to generate the SSH key-pair does not have the Nova client installed and openrc credential file, you will need to copy the public key (i.e. id_rsa.pub) to an OpenStack Control Node. The following command is from a host that contains the Nova client, openrc credential file and SSH public key:

```
nova keypair-add --pub-key id_rsa.pub <key_name>
```

Example:

```
nova keypair-add --pub-key /root/.ssh/id_rsa.pub test-key
```

This creates a new key-pair called test-key. The private key (id_rsa) is saved locally in ~/.ssh which can be used to connect to an instance launched using test-key as the Nova keypair name. You can see the available key-pairs with nova keypair-list command.

```
nova keypair-list
+-------+-------------------------------------------------+
|  Name |                   Fingerprint                   |
+-------+-------------------------------------------------+
| test-key  | b0:18:32:fa:4e:d4:3c:1b:c4:6c:dd:cb:53:29:13:82 |
+-------+-------------------------------------------------+
```

Boot an Instance and include the newly creatd Nova keypair:

```
nova boot --image cirros --flavor m1.tiny --key_name test-key --nic net-id=<neutron_net_id> <insta
```

The Instance should boot successfully. You can use the *nova show* <*instance_name*> command to obtain the IP address used to ping and SSH to the instance in the following command.
SSH to the Instance by specifing the -i and path to the SSH private key:

```
ssh -i /root/.ssh/id_rsa cirros@192.168.223.14

The authenticity of host '192.168.223.14 (192.168.223.14)' can't be established.
RSA key fingerprint is ef:62:fa:09:24:bd:67:b8:42:a5:2e:ce:c7:b1:65:41.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.223.14' (RSA) to the list of known hosts.
$
```

If you receive a Permissions are too open? private key will be ignored error, change the permissions of your private key so that it is only readable and writeable by the owner:

```
chmod 600 /root/.ssh/id_rsa
```

Configuring OpenStack Networking (Neutron) DHCP Agent High-Availability

First, verify the status of your Neutron 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:

```
neutron 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 Neutron public223 network:

```
neutron 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 neutron agent-list command.

```
neutron 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:~# neutron 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=<neutron_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 neutron agent-list command.

```
neutron 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:~# neutron 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=<neutron_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.

Create and Attach a Cinder Volume

You can use Horizon to create and attach Cinder volumes if you prefer. Since using Horizon to manage volumes is self-explanatory, this section covers the CLI for managing volumes. First, verify the status of your Cinder services from a controller. You should see :-) under State and enabled under Status for all services. Do not proceed with creating and attaching volumes if this is not the case:

```
cinder-manage service list
Binary           Host                                 Zone             Status     State Updated At
cinder-scheduler control01                            nova             enabled    :-)   2013-11-01
cinder-volume    compute01                            nova             enabled    :-)   2013-11-01
cinder-scheduler control02                            nova             enabled    :-)   2013-11-01
cinder-volume    compute02                            nova             enabled    :-)   2013-11-01
cinder-scheduler control03                            nova             enabled    :-)   2013-11-01
cinder-volume    compute03                            nova             enabled    :-)   2013-11-01
```

Create a Cinder volume. Our example uses *v2* for the volume name and is *2GB* in size:

```
cinder create --display-name v2 2
+---------------------+--------------------------------------+
|       Property      |                Value                 |
+---------------------+--------------------------------------+
|     attachments     |                  []                  |
|  availability_zone  |                 nova                 |
|       bootable      |                false                 |
|      created_at     |      2013-11-01T17:12:01.058239      |
| display_description |                 None                 |
|     display_name    |                  v2                  |
|          id         | 8b858ebe-c7aa-4674-af0d-7031bb5cc7a1 |
|       metadata      |                  {}                  |
|         size        |                  2                   |
|     snapshot_id     |                 None                 |
|     source_volid    |                 None                 |
|        status       |               creating               |
|     volume_type     |                 None                 |
+---------------------+--------------------------------------+
```

Attach the volume to a running instance. Our examples attaches the *8b858ebe-c7aa-4674-af0d-7031bb5cc7a1* volume (UUID of the v2 volume) to an Instance named *c2*.

```
nova volume-attach c2 8b858ebe-c7aa-4674-af0d-7031bb5cc7a1 /dev/vdb
```

```
+----------+--------------------------------------+
| Property | Value                                |
+----------+--------------------------------------+
| device   | /dev/vdvc                            |
| serverId | 91f1b1a3-f1b8-4efc-a3a2-8708447e9377 |
| id       | 8b858ebe-c7aa-4674-af0d-7031bb5cc7a1 |
| volumeId | 8b858ebe-c7aa-4674-af0d-7031bb5cc7a1 |
+----------+--------------------------------------+
```

Log into the c2 Instance and verify the volume attachment. You should observe a 2GB disk named /dev/vdb:

```
sudo -i
fdisk -l

Disk /dev/vda: 1073 MB, 1073741824 bytes
255 heads, 63 sectors/track, 130 cylinders, total 2097152 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x00000000

   Device Boot      Start         End      Blocks   Id  System
/dev/vda1   *       16065     2088449     1036192+  83  Linux

Disk /dev/vdb: 2147 MB, 2147483648 bytes
16 heads, 63 sectors/track, 4161 cylinders, total 4194304 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x00000000

Disk /dev/vdb doesn't contain a valid partition table
```

Partition the /dev/vdb disk:

```
fdisk /dev/vdb
```

Once in fdisk, perform the following commands:

1. Press **n** to create a new disk partition,
2. Press **p** to create a primary disk partition,
3. Press **1** to denote it as 1st disk partition,
4. Either press ENTER twice to accept the default of 1st and last cylinder ? to convert the remainder of hard disk to a single disk partition -OR- press ENTER once to accept the default of the 1st, and then choose how big you want the partition to be by specifying +size[K,M,G] e.g. +5G or +6700M.
5. Press **t** and select the new partition that you have created.
6. Press **83** change your new partition to 8e, i.e. Linux LVM partition type.
7. Press **p** to display the hard disk partition setup. Please take note that the first partition is denoted as /dev/sdb1 in Linux.
8. Press **w** to write the partition table and exit fdisk upon completion.

You should see your new partition named */dev/vdb1* in this listing.

```
fdisk -l

Disk /dev/vda: 1073 MB, 1073741824 bytes
255 heads, 63 sectors/track, 130 cylinders, total 2097152 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
```

```
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x00000000

   Device Boot      Start         End      Blocks   Id  System
/dev/vda1   *       16065     2088449     1036192+  83  Linux

Disk /dev/vdb: 2147 MB, 2147483648 bytes
1 heads, 16 sectors/track, 262144 cylinders, total 4194304 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x93bf0c66

   Device Boot      Start         End      Blocks   Id  System
/dev/vdb1            2048     4194303     2096128   83  Linux
```

Make a file system on the partition and mount it.

```
mkfs.ext3 /dev/vdb1
mkdir /cinder-test
mount /dev/vdb1 /cinder-test
```

In our experience, in order for the contents of a Cinder volume to become persistent, you must first reboot the Instance before writing data to the volume.

```
reboot now
```

Log back into the Instance, create a test file and reboot to test data persistence.

```
touch /cinder-test/test-file
reboot now
```

Single NIC Configurations

In the Networking Section, multiple NICs are used on Control and Compute Nodes. Eth0 is used for node management traffic and for associating API endpoints on Control Nodes. Eth1, which does not use an IP address, is used to bridge OVS traffic between instances and the provider's physical network. Below is an /etc/network/interfaces example (control01) for deployments requiring a single NIC on Control/Compute Nodes:

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

# Public Network: Bridged Interface
auto eth0
iface eth0 inet static
  address 192.168.220.41
  netmask 255.255.255.0
  gateway 192.168.220.1
  dns-nameservers 192.168.220.254
  dns-search dmz-pod2.lab
  post-down ip addr flush dev eth0

# Public Network: Bridged Interface
auto br-ex
iface br-ex inet manual
  pre-up ip link set eth0 down
  up ifconfig br-ex 192.168.220.41 netmask 255.255.255.0 up
  dns-search dmz-pod2.lab
```

```
dns-nameservers 192.168.220.254
up ip route add default via 192.168.220.1 dev br-ex
up ifconfig eth0 0.0.0.0 up
up ip link set eth0 up
post-up service openvswitch-switch restart
post-up service neutron-plugin-openvswitch-agent restart
post-up service neutron-dhcp-agent restart
post-up service nova-compute restart
post-down ip link set br-ex down
```

Cisco UCS Servers support NIC virtualization. With NIC virtualization, you can create multiple vNIC's per physical NIC. Therefore, you can create 2 vNIC's associated to physical NIC 0 and the Ubuntu Operating System will see eth0 and eth1. This will allow you to follow the same steps identified in the Networking Section, while only using a single physical NIC.

Support

Email: openstack-support@cisco.com

Credits

This work has been based on:

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

Authors

Daneyon Hansen