Contents

- 1 Background
- 2 High-Availability Introduction
- 3 Dependencies
  - 3.1 Critical Reminders
  - 3.2 Operating System
  - 3.3 Server Requirements
  - 3.4 Networking Requirements
- 4 Installation
  - 4.1 Build Node Installation
    - 4.1.1 Option 1: Run the Script
    - 4.1.2 Option 2: Run the Commands Manually
- 5 Build Node Configuration
- 6 Load-Balancer Node Deployment
  - 6.1 Verify the Load-Balancer Node Installation
- 7 Swift Storage Node Deployment
  - 7.1 Verify the Swift Storage Node Installation
- 8 Swift Proxy Node Deployment
  - 8.1 Verify the Proxy Node Installation
- 9 Controller Node Deployment
  - 9.1 Verify the Controller Node Installation
- 10 Compute Node Deployment
  - 10.1 Verify the Compute Node Installation
- 11 Deploy Your First Instance
- 12 Deploy Your First Volume
- 13 Configure Quantum DHCP Agent Redundancy
- 14 Support
- 15 Credits
- 16 Authors

Background

There are two common ways of installing OpenStack, either manually or by using automation tools such as the Cisco OpenStack Installer (Cisco OSI). This guide providers users step-by-step instructions for using Cisco OSI to automate the installation, configuration and deployment of a highly-available OpenStack Grizzly environment. The deployment covers the following OpenStack software components:

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

In addition to highly-available OpenStack services, the system will also enable basic monitoring functionality based on Nagios, Collectd, and Graphite.

In Cisco OSI, an initial build server outside of the OpenStack cluster is used to manage and automate the OpenStack software deployment. This build server primarily functions as a Puppet Master for software deployment and configuration management of the OpenStack cluster, as well as a Cobbler server for managing the underlying Ubuntu Operating System installation. Once the build server is installed and configured, it is used as an out-of-band automation and management workstation to manage the OpenStack nodes. It also functions as a monitoring server to collect statistics about the health and performance of the OpenStack deployment, as well as to monitor the availability of the machines and services.

You can find bugs, release milestones, and other information on Launchpad. You can find release notes for the most recent release here.

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 architecture consists of the following components used to provide high-availability to OpenStack services:

- MySQL Galera provides synchronous multi-master clustering to the MySQL/InnoDB databases.
- RabbitMQ Clustering and RabbitMQ Mirrored Queues provide active/active and highly scalable message queuing for OpenStack services.
- HAProxy and Keepalived provide load-balancing between clients and OpenStack API Endpoints.
- The Multiple Quantum L3 and DHCP Agents Blueprint allows multiple Quantum Layer-3 and DHCP Agents to be deployed for high-availability and scalability purposes. At this time (Grizzly release), multiple DHCP Agents can service a Quantum network, however, only a single L3 Agent can service one Quantum network at a time. Therefore, the L3 Agent is a single point of failure and is not included in the Cisco High-Availability Deployment Guide. Quantum Provider Network Extensions are used to map physical data center networks to Quantum networks. In this deployment model, Quantum relies on the physical data center to provide Layer-3 high-availability instead of the L3 Agent.
- Glance uses Swift as the back-end to store 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 proxies are used to provide high-availability for 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.

For more details about the high-availability architecture, please refer to the manual deployment guide

Dependencies

Critical Reminders

The most common OpenStack HA deployment issues are either incorrect site.pp file settings 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 files. You will likely be using your own IP addressing, passwords and node names 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 within the site.pp example file based on your specific deployment environment. Additional Swift considerations and tuning information can be found here.

The passwords used in our example site.pp file are insecure, so it's highly recommended to change them.

Operating System

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

Server Requirements

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

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

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

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

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

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

Networking Requirements

The OpenStack HA environment uses five separate networks. Three of the five networks are used by Tenants. Three tenant networks are being used as an example, and thus the tenant networks can be increased or decreased based on your deployment needs. Connectivity within Tenants uses Quantum with the Open vSwitch (OVS) plugin and Provider Network Extensions. Provider Network Extensions allow cloud

administrators to create OpenStack networks that map directly to physical networks in the data center and support local, VLAN and GRE deployment models. Our example uses the Provider VLAN networking model. The network details are as follows:

- **1 Management Network**
  - This network is used to perform management functions against the node. For example, SSH'ing to the nodes to change a configuration setting. The network is also used for lights-out management using the CIMC interface of the UCS servers. Lastly, OpenStack API's and the Horizon web dashboard is associated to this network.
  - An IP address for each node is required for this network. If using lights-out management such as CIMC, each node will require 2 addresses from this network.
  - This network typically employs private (RFC1918).
- **3 Tenant Networks**
  - These networks are used to provide connectivity to Instances. Since Quantum Provider Networking Extensions are being used, it is common to give tenants direct access to a "public" network that can be used to reach the Internet.
  - Compute and Controller Nodes will have an interface attached to this network. Since the Compute/Controller Node interfaces that attach to this network are managed by OVS, the interface 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 (this document uses all private addressing).
- **1 Storage Network**
  - This network is used for providing separate connectivity between Swift Proxy and Storage Nodes. This ensures storage traffic is not interfering with Instance traffic.
  - This network typically employs private (RFC1918) IP addressing.

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

Figure 1: OpenStack HA Network Design Details

.

- **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 Quantum Provider Networking

Extensions, so tenant networks will also use ASR 9000 VRRP addresses as gateways.

Installation

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

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

Build Node Installation

Ensure you have reviewed the Critical Reminders section before proceeding. This server has relatively modest hardware requirements: 2 GB RAM, 20 GB storage, Internet connectivity, and a network interface on the same network as the management interfaces (CIMC and eth0 in our reference deployment) of the OpenStack nodes.

Install Ubuntu 12.04 LTS. A minimal install with openssh-server is sufficient. Configure the network interface on the OpenStack cluster management segment with a static IP. Also, when partitioning the storage, choose a partitioning scheme which provides at least 15 GB free space under /var, as installation packages and ISO images used to deploy OpenStack will eventually be cached there. Use the Build Node Deployment Guide for step-by-step instructions for installing the Build Node. When the installation finishes, log in and become root:

```
sudo -H bash
```

NOTE: Please read the following if you have proxy'ed Internet access or no Internet access :

If you require a proxy server to access the Internet, be aware that proxy users have occasionally reported problems during the phases of the installation process that download and install software packages. A common symptom of proxy trouble is that apt will complain about hash mismatches or file corruptions when verifying downloaded files. A few known scenarios and workarounds include:

- If the apt-get process reports a "HASH mismatch", you may be facing an issue with a caching engine. If it's possible to do so, bypassing the caching engine may resolve the problem.
- If you do have a proxy, you will want, at a minimum, to export the two types of proxies needed in your root shell when running fetch commands, as noted in the relevant sections.
- You will also want to change the $proxy setting in site.pp to reflect your local proxy.

Another possible change is if you don't have "public" Internet accessible IPs for all of your machines (build, control, compute, etc.) and are building this in a controlled environment. If this is the case, ensure that $default_gateway is *not* set in site.pp and all of the files required for installing the control and compute nodes will be fetched from the boot server.

IMPORTANT: If you have proxies, and you set your proxy information in either your .profile or in a file like /etc/environment, you will need to set both http_proxy and https_proxy. You will also need to set a no_proxy command at least for the build node. An example might look like:

```
http_proxy=http://your-proxy.address.com:80/
https_proxy=https://your-https-proxy.address.com:443/
```

```
no_proxy=your-build-node-name,*yourbuild.domain.name,127.0.0.1,127.0.1.1,localhost
```

You have two choices for configuring up the build node. You can follow the manual steps below, or you can run a one line script that tries to automate this process. In either case, you should end up with the puppet modules installed, and a set of template site manifests in /etc/puppet/manifests.

Option 1: Run the Script

To run the install script, copy and paste the following on your command line (as root with your proxy set if necessary as above):

```
curl -s -k -B https://raw.github.com/CiscoSystems/grizzly-manifests/multi-node/install_os_puppet |
```

With a proxy, use:

```
https_proxy=http://proxy.example.com:80/ curl -s -k -B https://raw.github.com/CiscoSystems/grizzly
chmod +x install_os_puppet
./install_os_puppet -p http://proxy.example.com:80/
```

You can now jump to "Customizing your build server". Otherwise, follow along with the steps below.

Option 2: Run the Commands Manually

All should now install any pending security updates:

```
apt-get update && apt-get dist-upgrade -y && apt-get install -y puppet git ipmitool
```

Reboot the system:

```
reboot now
```

Get the Cisco OSI example manifests. Under the grizzly-manifests GitHub repository:

```
git clone -b multi-node https://github.com/CiscoSystems/grizzly-manifests ~/cisco-grizzly-manifest
cd ~/cisco-grizzly-manifests
git checkout -q g.2
```

With a proxy:

```
https_proxy=http://proxy.example.com:80 git clone -b multi-node https://github.com/CiscoSystems/gr
cd ~/cisco-grizzly-manifests
https_proxy=http://proxy.example.com:80 git checkout multi-node
```

Copy the puppet manifests from ~/cisco-grizzly-manifests/manifests/ to /etc/puppet/manifests/

```
cp ~/cisco-grizzly-manifests/manifests/* /etc/puppet/manifests
```

Copy the puppet templates from ~cisco-grizzly-manifests/templates/ to /etc/puppet/templates/

```
cp ~/cisco-grizzly-manifests/templates/* /etc/puppet/templates
```

Then get the Cisco Edition puppet modules from Cisco's GitHub repository:

```
(cd /etc/puppet/manifests; python /etc/puppet/manifests/puppet_modules.py)
```

With a proxy:

```
(cd /etc/puppet/manifests; http_proxy=http://proxy.example.com:80 https_proxy=http://proxy.example
```

Build Node Configuration

In the /etc/puppet/manifests directory you will find these files:

```
clean_node.sh
cobbler-node.pp
core.pp
modules.list
puppet-modules.py
reset_nodes.sh
site.pp.example
site.pp.ha.example
```

At a high level, cobbler-node.pp manages the deployment of cobbler to support booting of additional servers into your environment. The core.pp manifest defines the core definitions for OpenStack service deployment. The site.pp.example manifest captures the configuration components that can be modified by end-users ""for a non-HA deployment"". The site.pp.ha.example manifest captures the configuration components that can be modified by end-users ""for a HA deployment"". clean_node.sh is a shell script provided as a convenience to deployment users; it wraps several cobbler and puppet commands for ease of use when building and rebuilding the nodes of the OpenStack cluster. reset_nodes.sh is a wrapper around clean_node.sh to rebuild your entire cluster quickly with one command.

IMPORTANT! You must copy site.pp.ha.example to site.pp and then edit it as appropriate for your installation. It is internally documented.

```
cp /etc/puppet/manifests/site.pp.ha.example /etc/puppet/manifests/site.pp
vi /etc/puppet/manifests/site.pp
```

Then, use the ?puppet apply? command to activate the manifest:

```
puppet apply -v /etc/puppet/manifests/site.pp
```

When the puppet apply command runs, the puppet agent on the build node will follow the instructions in the site.pp and cobbler-node.pp manifests and will configure several services on the build server:

- ntpd -- a time synchronization server used on all OpenStack cluster nodes to ensure time throughout the cluster is correct
- tftpd-hpa -- a TFTP server used as part of the PXE boot process when OpenStack nodes boot up
- dnsmasq -- a DNS and DHCP server used as part of the PXE boot process when OpenStack nodes boot up
- cobbler -- an installation and boot management daemon which manages the installation and booting of OpenStack nodes
- apt-cacher-ng -- a caching proxy for package installations, used to speed up package installation on the OpenStack nodes
- nagios -- a infrastructure monitoring application, used to monitor the servers and processes of the OpenStack cluster
- collectd --a statistics collection application, used to gather performance and other metrics from the components of the OpenStack cluster
- graphite and carbon -- a real-time graphing system for parsing and displaying metrics and statistics about OpenStack
- apache -- a web server hosting sites to implement graphite, nagios, and puppet web services

The initial puppet configuration of the build node will take several minutes to complete as it downloads, installs, and configures all the software needed for these applications.

Once the site manifest has been applied to your system, you need to stage puppet plugins so they can be accessed by the managed nodes:

```
puppet plugin download
```

After the build node is configured, the systems listed in site.pp should be defined in cobbler on the build server:

```
cobbler system list
   slb01
   slb02
   control01
   control02
   control03
   swiftproxy01
   swiftproxy02
   swift01
   swift02
   swift03
   compute01
   compute02
```

Load-Balancer Node Deployment

Now that the OpenStack nodes are being managed by the Build Node, deploying a node should be as simple as calling the clean_node.sh script with the node name:

```
. /etc/puppet/manifests/clean_node.sh {node_name}
```

NOTE: Replace node_name with the name of your controller.

```
Example:
/etc/puppet/manifests/clean_node.sh slb01
/etc/puppet/manifests/clean_node.sh slb02
```

clean_node.sh is a script which does several things:

- Configures Cobbler to PXE boot the specified node with appropriate PXE options to do an automated install of Ubuntu
- Uses Cobbler to power-cycle the node
- Removes any existing client registrations for the node from Puppet, so Puppet will treat it as a new install
- Removes any existing key entries for the node from the SSH known hosts database

You can watch the automated Ubuntu installation progress by using the Virtual KVM from the UCS CIMC. Once the installation finishes, nodes reboot and then automatically run the puppet agent. By now you should be able to SSH to the node(s):

```
 ssh localadmin@<MGT_IP>
```

On each node, the Puppet agent automatically retrieves and applies the configuration associated to its name within the site manifest. This step will take several minutes, as puppet downloads, installs, and configures the various OpenStack components and associated services. Observe the progress of the puppet deployment process by tail'ing syslog:

```
tail -f /var/log/syslog
```

Once OpenStack nodes complete the build process, run puppet on the build node a second time:

```
puppet agent -t
```

This second puppet run will gather information about the individual OpenStack nodes collected by Puppet when they were being built, and use that information to set up status monitoring of the OpenStack cluster on the build server.

Verify the Load-Balancer Node Installation

After the load-balancer nodes complete the automated deployment process, use the following steps to verify proper functionality:

Make sure puppet agent runs clean:

```
puppet agent -t -d
```

Note: If you are unable to complete a clean puppet agent run, then do not proceed with the deployment. Send a support request email to openstack-support@cisco.com with a copy of the puppet error.

Verify that the haproxy service is running:

```
service haproxy status
haproxy is running.
```

Verify that the virtual IP address for the Controller cluster is bound to node slb01:

```
ip addr list

1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether f8:72:ea:00:1c:4d brd ff:ff:ff:ff:ff:ff
    inet 192.168.220.81/24 brd 192.168.220.255 scope global eth0
    ""inet 192.168.220.40/32 scope global eth0""
    inet6 fe80::fa72:eaff:fe00:1c4d/64 scope link
       valid_lft forever preferred_lft forever
```

Verify that the virtual IP address for the swift proxy cluster is bound to node slb02:

```
ip addr list

1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
```

```
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether f8:72:ea:00:1c:4d brd ff:ff:ff:ff:ff:ff
    inet 192.168.220.82/24 brd 192.168.220.255 scope global eth0
    ""inet 192.168.220.60/32 scope global eth0""
    inet6 fe80::fa72:eaff:fe00:1c4d/64 scope link
       valid_lft forever preferred_lft forever
```

Swift Storage Node Deployment

Now that the OpenStack nodes are being managed by the Build Node, deploying Swift storage nodes should be as simple as calling the clean_node.sh script with the node name. You can deploy all the storage nodes at the same time.

```
. /etc/puppet/manifests/clean_node.sh {node_name}
```

Note: Replace node_name with the name of your Swift storage nodes. Example:

```
/etc/puppet/manifests/clean_node.sh swift01
/etc/puppet/manifests/clean_node.sh swift02
/etc/puppet/manifests/clean_node.sh swift03
```

Login to the storage nodes after the Ubuntu provisioning process is complete and view the results of your puppet agent run:

```
tail -f /var/log/syslog
```

Your puppet agent run on storage nodes should complete with the following errors:

```
err: /Service[swift-account-replicator]/ensure: change from stopped to running failed: Could not s

err: /Service[swift-object-replicator]/ensure: change from stopped to running failed: Could not st

err: /Service[swift-container-replicator]/ensure: change from stopped to running failed: Could not

err: /Service[swift-container-sync]/ensure: change from stopped to running failed: Could not start
```

The errors occur because these services rely on the ring files (account.ring.gz, container.ring.gz, and object.ring.gz) that get created on the proxy nodes and then get rsync'ed over to the storage nodes (on the 2nd puppet agent run). If you receive any other errors, try running the puppet agent again. If you are unable to complete a puppet agent run with only the above mentioned errors, do not proceed with the deployment and send an email support request to openstack-support@cisco.com.

Once the Swift storage Nodes complete the build process, run puppet on the build node a second time:

```
puppet agent -t
```

Verify the Swift Storage Node Installation

After the storage nodes complete the automated deployment process, use the following steps to verify proper functionality:

Make sure puppet agent runs clean:

```
puppet agent -t -d
```

Note: If you are unable to complete a clean puppet agent run, then do not proceed with the deployment. Send a support request email to openstack-support@cisco.com with a copy of the puppet run error.

From each of the storage nodes, verify that the object/container/account storage services are running:

```
root@swift01:~# service swift-object status
swift-object start/running

root@swift01:~# service swift-container status
swift-container start/running

root@swift01:~# service swift-account status
swift-account start/running
```

Verify that the hard disks are partitioned and mounted. **Note:** Our example uses 5 hard disks named /dev/sdb-sdf (/dev/sda is used for the OS installation and not for Swift storage).

```
root@swift01:~# cat /etc/fstab
# <file system> <mount point>   <type>  <options>       <dump>  <pass>
proc    /proc   proc    nodev,noexec,nosuid     0       0
/dev/mapper/cinder--volumes-slash       /       ext4    errors=remount-ro       0       1
# /boot was on /dev/sdc2 during installation
UUID=828477c6-06f6-4828-98c3-9da9df11d604       /boot   ext3    defaults        0       2
/dev/mapper/cinder--volumes-var /var    ext4    defaults        0       2
/dev/mapper/cinder--volumes-swap        none    swap    sw      0       0
/dev/sdb        /srv/node/sdb   xfs     noatime,nodiratime,nobarrier,logbufs=8  0       0
/dev/sdf        /srv/node/sdf   xfs     noatime,nodiratime,nobarrier,logbufs=8  0       0
/dev/sde        /srv/node/sde   xfs     noatime,nodiratime,nobarrier,logbufs=8  0       0
/dev/sdc        /srv/node/sdc   xfs     noatime,nodiratime,nobarrier,logbufs=8  0       0
/dev/sdd        /srv/node/sdd   xfs     noatime,nodiratime,nobarrier,logbufs=8  0       0

root@swift01:~# cat /etc/mtab
/dev/mapper/cinder--volumes-slash / ext4 rw,errors=remount-ro 0 0
proc /proc proc rw,noexec,nosuid,nodev 0 0
sysfs /sys sysfs rw,noexec,nosuid,nodev 0 0
none /sys/fs/fuse/connections fusectl rw 0 0
none /sys/kernel/debug debugfs rw 0 0
none /sys/kernel/security securityfs rw 0 0
udev /dev devtmpfs rw,mode=0755 0 0
devpts /dev/pts devpts rw,noexec,nosuid,gid=5,mode=0620 0 0
tmpfs /run tmpfs rw,noexec,nosuid,size=10%,mode=0755 0 0
none /run/lock tmpfs rw,noexec,nosuid,nodev,size=5242880 0 0
none /run/shm tmpfs rw,nosuid,nodev 0 0
/dev/sdb /srv/node/sdb xfs rw,noatime,nodiratime,nobarrier,logbufs=8 0 0
/dev/sdf /srv/node/sdf xfs rw,noatime,nodiratime,nobarrier,logbufs=8 0 0
/dev/sde /srv/node/sde xfs rw,noatime,nodiratime,nobarrier,logbufs=8 0 0
/dev/sdc /srv/node/sdc xfs rw,noatime,nodiratime,nobarrier,logbufs=8 0 0
/dev/sdd /srv/node/sdd xfs rw,noatime,nodiratime,nobarrier,logbufs=8 0 0
/dev/sda2 /boot ext3 rw 0 0
/dev/mapper/cinder--volumes-var /var ext4 rw 0 0
```

Make sure to complete the same verification steps on all storage nodes. Once you have verified the deployment of all your storage nodes, proceed to deploying your first Swift proxy node.

Swift Proxy Node Deployment

Now that the OpenStack nodes are being managed by the Build Node, deploying Swift proxy nodes should be as simple as calling the clean_node.sh script with the node name. Make sure you deploy the first proxy node and verify the proper operation of the Swift rings before deploying the second proxy node.

```
. /etc/puppet/manifests/clean_node.sh {node_name}
```

Note: Replace node_name with the name of your first Swift proxy node. Example:

```
/etc/puppet/manifests/clean_node.sh swiftproxy01
```

On the proxy node, verify the existence of the Swift ring files:

```
ls -l /etc/swift/*.gz

-rw-r--r-- 1 root swift 466440 Sep  3 05:39 /etc/swift/account.ring.gz
-rw-r--r-- 1 root swift 466739 Sep  3 05:38 /etc/swift/container.ring.gz
-rw-r--r-- 1 root swift 466500 Sep  3 05:38 /etc/swift/object.ring.gz
```

Now that the ring files have been created on your first Swift proxy node, go back to your storage nodes and run puppet to copy the ring files needed to start the container-replicator, object-replicator, account-replicator, and container-sync services:

```
pupet agent -t -d
```

The puppet agent run should now complete without any errors.

On the storage nodes, verify the existence of the Swift ring files:

```
ls -l /etc/swift/*.gz

-rw-r--r-- 1 root swift 466440 Sep  3 05:39 /etc/swift/account.ring.gz
-rw-r--r-- 1 root swift 466739 Sep  3 05:38 /etc/swift/container.ring.gz
-rw-r--r-- 1 root swift 466500 Sep  3 05:38 /etc/swift/object.ring.gz
```

Once the Swift proxy node completes the build process, run puppet on the build node a second time:

```
puppet agent -t
```

Verify the Proxy Node Installation

After the first proxy node completes the automated deployment process, use the following steps to verify proper functionality:

Make sure puppet agent runs clean:

```
puppet agent -t -d
```

Note: If you are unable to complete a clean puppet agent run, then do not proceed with the deployment. Send a support request email to openstack-support@cisco.com with a copy of the puppet run error.

From the proxy node, verify that the proxy service is running:

```
root@swiftproxy01:~# service swift-proxy status
swift-proxy start/running
```

Verify that all the devices across all storage nodes appear in each of the 3 Swift rings. **Note:** Our example uses 3 storage nodes and 5 hard disks named /dev/sdb-sdf (/dev/sda is used for the OS installation and not for Swift storage).

```
root@swiftproxy01:~# 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, 0.00 balance
The minimum number of hours before a partition can be reassigned is 1
Devices:    id  region  zone      ip address  port      name weight partitions balance meta
             0     1     3  192.168.222.73  6002       sde   1.00      52429   0.00
             1     1     2  192.168.222.72  6002       sdd   1.00      52428  -0.00
             2     1     3  192.168.222.73  6002       sdc   1.00      52429   0.00
             3     1     2  192.168.222.72  6002       sdb   1.00      52429   0.00
             4     1     3  192.168.222.73  6002       sdb   1.00      52429   0.00
             5     1     1  192.168.222.71  6002       sdb   1.00      52429   0.00
             6     1     1  192.168.222.71  6002       sdc   1.00      52429   0.00
             7     1     2  192.168.222.72  6002       sdf   1.00      52429   0.00
             8     1     1  192.168.222.71  6002       sdd   1.00      52429   0.00
             9     1     2  192.168.222.72  6002       sdc   1.00      52429   0.00
            10     1     1  192.168.222.71  6002       sde   1.00      52428  -0.00
            11     1     1  192.168.222.71  6002       sdf   1.00      52429   0.00
            12     1     3  192.168.222.73  6002       sdf   1.00      52429   0.00
            13     1     2  192.168.222.72  6002       sde   1.00      52429   0.00
            14     1     3  192.168.222.73  6002       sdd   1.00      52428  -0.00

root@swiftproxy01:~# 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, 0.00 balance
The minimum number of hours before a partition can be reassigned is 1
Devices:    id  region  zone      ip address  port      name weight partitions balance meta
             0     1     2  192.168.222.72  6001       sdb   1.00      52429   0.00
             1     1     1  192.168.222.71  6001       sdc   1.00      52429   0.00
             2     1     3  192.168.222.73  6001       sdf   1.00      52429   0.00
             3     1     3  192.168.222.73  6001       sdb   1.00      52429   0.00
             4     1     3  192.168.222.73  6001       sde   1.00      52429   0.00
             5     1     1  192.168.222.71  6001       sdb   1.00      52429   0.00
             6     1     2  192.168.222.72  6001       sdc   1.00      52429   0.00
             7     1     1  192.168.222.71  6001       sdf   1.00      52429   0.00
             8     1     2  192.168.222.72  6001       sdf   1.00      52429   0.00
             9     1     1  192.168.222.71  6001       sdd   1.00      52429   0.00
            10     1     1  192.168.222.71  6001       sde   1.00      52428  -0.00
            11     1     2  192.168.222.72  6001       sde   1.00      52429   0.00
            12     1     2  192.168.222.72  6001       sdd   1.00      52428  -0.00
            13     1     3  192.168.222.73  6001       sdd   1.00      52429   0.00
            14     1     3  192.168.222.73  6001       sdc   1.00      52428  -0.00

root@swiftproxy01:~# 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, 0.00 balance
The minimum number of hours before a partition can be reassigned is 1
Devices:    id  region  zone      ip address  port      name weight partitions balance meta
             0     1     1  192.168.222.71  6000       sdc   1.00      52429   0.00
             1     1     2  192.168.222.72  6000       sdb   1.00      52429   0.00
             2     1     3  192.168.222.73  6000       sdf   1.00      52429   0.00
             3     1     1  192.168.222.71  6000       sdf   1.00      52428  -0.00
             4     1     2  192.168.222.72  6000       sde   1.00      52429   0.00
             5     1     3  192.168.222.73  6000       sdc   1.00      52429   0.00
             6     1     2  192.168.222.72  6000       sdd   1.00      52429   0.00
             7     1     1  192.168.222.71  6000       sde   1.00      52429   0.00
             8     1     3  192.168.222.73  6000       sdd   1.00      52429   0.00
             9     1     2  192.168.222.72  6000       sdf   1.00      52429   0.00
            10     1     3  192.168.222.73  6000       sdb   1.00      52429   0.00
            11     1     1  192.168.222.71  6000       sdb   1.00      52429   0.00
            12     1     3  192.168.222.73  6000       sde   1.00      52428  -0.00
            13     1     2  192.168.222.72  6000       sdc   1.00      52428  -0.00
            14     1     1  192.168.222.71  6000       sdd   1.00      52429   0.00
```

Verify that the ring files are present on your storage nodes:

```
ls -l /etc/swift/

-rw-r--r-- 1 root  swift 432702 Sep  5 17:22 account.ring.gz
drwxr-sr-x 2 swift swift   4096 Sep  5 17:09 account-server
-rw-r----- 1 swift swift    455 Sep  5 17:09 account-server.conf
-rw-r--r-- 1 root  swift 432511 Sep  5 17:21 container.ring.gz
drwxr-sr-x 2 swift swift   4096 Sep  5 17:09 container-server
-rw-r----- 1 swift swift    541 Sep  5 17:09 container-server.conf
-rw-r--r-- 1 root  swift 432729 Sep  5 17:22 object.ring.gz
drwxr-sr-x 2 swift swift   4096 Sep  5 17:09 object-server
-rw-r----- 1 swift swift    448 Sep  5 17:09 object-server.conf
-rw-rw---- 1 swift swift     51 Sep  5 17:09 swift.conf
```

Verify that your Swift storage services are running on your storage nodes. If not, try using the swift-init all start command to manually start the services. On storage nodes, all services except proxy-server and object-expirer should be running.

```
swift-init all status

container-updater running (11566 - /etc/swift/container-server.conf)
account-auditor running (11567 - /etc/swift/account-server.conf)
object-replicator running (9696 - /etc/swift/object-server.conf)
No proxy-server running
container-replicator running (11568 - /etc/swift/container-server.conf)
object-auditor running (11569 - /etc/swift/object-server.conf)
No object-expirer running
container-auditor running (11570 - /etc/swift/container-server.conf)
container-server running (11571 - /etc/swift/container-server.conf)
account-server running (11572 - /etc/swift/account-server.conf)
account-reaper running (11573 - /etc/swift/account-server.conf)
container-sync running (11574 - /etc/swift/container-server.conf)
account-replicator running (11576 - /etc/swift/account-server.conf)
object-updater running (11590 - /etc/swift/object-server.conf)
object-server running (11594 - /etc/swift/object-server.conf)
```

After the first proxy node and all storage nodes are properly deployed, follow the same steps to deploy the second proxy node. **Note:** After the Swift rings have been created, additional puppet agent runs on storage nodes will not work due to the following error:

```
err: Could not retrieve catalog from remote server: Error 400 on SERVER: Exported resource Swift::
warning: Not using cache on failed catalog
err: Could not retrieve catalog; skipping run
```

Controller Node Deployment

Now that the OpenStack nodes are being managed by the Build Node, deploying the controller nodes should be as simple as calling the clean_node.sh script with the node name. It is **IMPORTANT** to deploy the controller nodes one at a time starting with control01, then control02 and lastly control03

```
/etc/puppet/manifests/clean_node.sh {node_name}
```

Note: Replace node_name with the name of your first controller node. Example:

```
/etc/puppet/manifests/clean_node.sh control01
```

Once control01 completes the build process, run puppet on the build node a second time:

```
puppet agent -t
```

This second puppet run will gather information about the individual OpenStack nodes collected by puppet when they were being built, and use that information to set up status monitoring of the OpenStack cluster on the build server.

Note: It is important to understand the gcomm address concept behind Galera database clustering technology used within the OpenStack High Availability architecture. The gcomm address is represented as galera_master_ip in the Puppet site.pp.ha.example manifest. Only use an empty gcomm:// address (i.e. galera_master_ip => false) when you create a NEW cluster. Never use it when your intention is to reconnect to an existing cluster. This is why the control01 node definition in site.pp.ha.example has a second, commented-out, galera_master_ip definition of control02. After the Galera cluster is established, you should change the galera_master_ip => false to $controller02_ip, or $controller03_ip. Otherwise, control01 will not join the cluster upon reboot. It is equally important to understand how to restart an existing Galera cluster in the event of a power outage.

Verify the Controller Node Installation

After the controller nodes complete the automated deployment process, use the following steps to verify proper functionality:

Make sure puppet agent runs clean:

```
puppet agent -t -d
```

Note: If you are unable to complete a clean puppet agent run, then do not proceed with the deployment. Send a support request email to openstack-support@cisco.com with a copy of the puppet run error.

From each of the load-balancer nodes, verify that the MySQL Galera monitoring service is functioning. If the curl output does not show 'MySQL is running', then rerun the puppet agent on the associated control node and run the curl command again. If you are unable to get MySQL to run, do not proceed with the deployment. Send a support request email to openstack-support@cisco.com with a copy of the associated puppet run error.

```
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.
```

Verify that the RabbitMQ service is running:

```
service rabbitmq-server status
```

Verify that the OpenStack services are running:

```
service keystone status
service glance-api status
service glance-registry status
```

```
service nova-api status
service nova-conductor status
service nova-scheduler status
service nova-consoleauth status
service quantum-server status
service quantum-dhcp-agent status
service quantum-plugin-openvswitch status
service cinder-api status
service cinder-scheduler status
```

You can also use the OpenStack Clients to verify the status of Nova and Quantum services. First, you need to source your authentication credentials:

```
source openrc
```

Verify the status of Nova services:

```
root@control01:~# nova-manage service list
Binary           Host                                 Zone             Status     State Updated_At
nova-consoleauth control01                            internal         enabled    :-)   2013-09-03
nova-scheduler   control01                            internal         enabled    :-)   2013-09-03
nova-conductor   control01                            internal         enabled    :-)   2013-09-03
nova-cert        control01                            internal         enabled    :-)   2013-09-03
```

Verify the status of Quantum agents:

```
root@control01:~# quantum agent-list
+--------------------------------------+--------------------+------------------------+-------+----
| id                                   | agent_type         | host                   | alive | adm
+--------------------------------------+--------------------+------------------------+-------+----
| 902947a4-a678-46d3-b0d7-e42060a9c096 | DHCP agent         | control01.dmz-pod2.lab | :-)   | Tru
| a08605a8-5182-444a-b9d8-bc6dca02cc00 | Open vSwitch agent | control01.dmz-pod2.lab | :-)   | Tru
+--------------------------------------+--------------------+------------------------+-------+----
```

Verify that you can obtain a token from Keystone:

```
keystone token-get
```

Now that the first controller is properly deployed, follow the same steps for the other two controller nodes (one at a time). After all 3 controller nodes have been successfully deployed, use the following steps to verify the proper operation of the controller cluster:

Verify the MySQL Galera cluster. Make sure the **wsrep_cluster_size** shows 3, the **wsrep_local_state_comment** shows Synced and **wsrep_connected/wsrep_ready** ON:

```
root@control01:~# mysql -e "show status like 'wsrep%';"

+----------------------------+-------------------------------------------------------------+
| Variable_name              | Value                                                       |
+----------------------------+-------------------------------------------------------------+
| wsrep_local_state_uuid     | 444c6327-1460-11e3-0800-75a812e3ed5f                        |
| wsrep_protocol_version     | 4                                                           |
| wsrep_last_committed       | 47236                                                       |
| wsrep_replicated           | 45291                                                       |
| wsrep_replicated_bytes     | 22317919                                                    |
| wsrep_received             | 2028                                                        |
| wsrep_received_bytes       | 822819                                                      |
| wsrep_local_commits        | 44709                                                       |
| wsrep_local_cert_failures  | 1                                                           |
| wsrep_local_bf_aborts      | 0                                                           |
```

```
| wsrep_local_replays        | 0                                                                |
| wsrep_local_send_queue     | 0                                                                |
| wsrep_local_send_queue_avg | 0.000000                                                         |
| wsrep_local_recv_queue     | 0                                                                |
| wsrep_local_recv_queue_avg | 0.000000                                                         |
| wsrep_flow_control_paused  | 0.000000                                                         |
| wsrep_flow_control_sent    | 0                                                                |
| wsrep_flow_control_recv    | 0                                                                |
| wsrep_cert_deps_distance   | 29.156863                                                        |
| wsrep_apply_oooe           | 0.000000                                                         |
| wsrep_apply_oool           | 0.000000                                                         |
| wsrep_apply_window         | 1.000000                                                         |
| wsrep_commit_oooe          | 0.000000                                                         |
| wsrep_commit_oool          | 0.000000                                                         |
| wsrep_commit_window        | 1.000000                                                         |
| wsrep_local_state          | 4                                                                |
| wsrep_local_state_comment  | Synced                                                           |
| wsrep_cert_index_size      | 37                                                               |
| wsrep_causal_reads         | 0                                                                |
| wsrep_incoming_addresses   | 192.168.220.41:3306,192.168.220.42:3306,192.168.220.43:3306 |
| wsrep_cluster_conf_id      | 11                                                               |
| wsrep_cluster_size         | 3                                                                |
| wsrep_cluster_state_uuid   | 444c6327-1460-11e3-0800-75a812e3ed5f                             |
| wsrep_cluster_status       | Primary                                                          |
| wsrep_connected            | ON                                                               |
| wsrep_local_index          | 0                                                                |
| wsrep_provider_name        | Galera                                                           |
| wsrep_provider_vendor      | Codership Oy <info@codership.com>                                |
| wsrep_provider_version     | 23.2.2(r137)                                                     |
| wsrep_ready                | ON                                                               |
+----------------------------+------------------------------------------------------------------+
```

Verify the RabbitMQ cluster shows all 3 controller nodes:

```
root@control01:~# rabbitmqctl cluster_status

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

Verify that you can talk to the Swift object storage service:

```
root@control01:~# swift -V 2 -A http://192.168.220.40:5000/v2.0/ -V 2 -U services:swift -K swift_p

   Account: AUTH_ce9c18b8066445459e079bec7df9f88b
Containers: 0
   Objects: 0
     Bytes: 0
Accept-Ranges: bytes
X-Timestamp: 1378231745.19399
X-Trans-Id: txd5ec450cefe64047a705dc24ce3349fb
Content-Type: text/plain; charset=utf-8
```

You can also use the Nova Client commands to verify the status of Nova services (minus Nova API) across all controller nodes:

```
root@control01:~# nova-manage service list

Binary           Host                                 Zone             Status     State Updated_At
nova-consoleauth control01                            internal         enabled    :-)   2013-09-03
nova-scheduler   control01                            internal         enabled    :-)   2013-09-03
```

```
nova-conductor   control01                          internal         enabled    :-)   2013-09-03
nova-cert        control01                          internal         enabled    :-)   2013-09-03
nova-consoleauth control02                          internal         enabled    :-)   2013-09-03
nova-scheduler   control02                          internal         enabled    :-)   2013-09-03
nova-conductor   control02                          internal         enabled    :-)   2013-09-03
nova-cert        control02                          internal         enabled    :-)   2013-09-03
nova-consoleauth control03                          internal         enabled    :-)   2013-09-03
nova-scheduler   control03                          internal         enabled    :-)   2013-09-03
nova-conductor   control03                          internal         enabled    :-)   2013-09-03
nova-cert        control03                          internal         enabled    :-)   2013-09-03
```

You can also use the Quantum Client commands to verify the status of Quantum Agents across all controller nodes:

```
root@control01:~# quantum agent-list

+--------------------------------------+--------------------+------------------------+-------+----
| id                                   | agent_type         | host                   | alive | adm
+--------------------------------------+--------------------+------------------------+-------+----
| 66c55a41-d66c-4100-a7ed-ecd2add48100 | Open vSwitch agent | control02.dmz-pod2.lab | :-)   | Tru
| 8bd0f911-6a6f-4735-8831-f967a5e792a7 | DHCP agent         | control03.dmz-pod2.lab | :-)   | Tru
| 902947a4-a678-46d3-b0d7-e42060a9c096 | DHCP agent         | control01.dmz-pod2.lab | :-)   | Tru
| 9519c99a-6061-4b72-866c-9b23046e3a20 | Open vSwitch agent | control03.dmz-pod2.lab | :-)   | Tru
| a08605a8-5182-444a-b9d8-bc6dca02cc00 | Open vSwitch agent | control01.dmz-pod2.lab | :-)   | Tru
| c2c2b349-465e-4c04-9285-14005cbba3be | DHCP agent         | control02.dmz-pod2.lab | :-)   | Tru
+--------------------------------------+--------------------+------------------------+-------+----
```

Verify that the Horizon web dashboard is operational. In our example deployment, **admin/Cisco123** are the default authentication credentials. Open a browser window and enter the following URL:

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

Compute Node Deployment

Now that the OpenStack nodes are being managed by the Build Node, deploying the compute nodes should be as simple as calling the clean_node.sh script with the node name. Unlike controller nodes, compute nodes can be deployed all at once.

```
/etc/puppet/manifests/clean_node.sh {node_name}
```

Note: Replace node_name with the name of your compute nodes. Example:

```
/etc/puppet/manifests/clean_node.sh compute01
/etc/puppet/manifests/clean_node.sh compute02
/etc/puppet/manifests/clean_node.sh compute03
```

Once the compute nodes complete the build process, run puppet on the build node a second time:

```
puppet agent -t
```

Verify the Compute Node Installation

After compute nodes complete their automated deployment process, use the following steps to verify proper functionality:

Make sure puppet agent runs clean:

```
puppet agent -t -d
```

Note: If you are unable to complete a clean puppet agent run, then do not proceed with the deployment. Send a support request email to openstack-support@cisco.com with a copy of the puppet run error.

Make sure OpenStack services are running:

```
root@compute02:~# service nova-compute status
nova-compute start/running, process 19261

root@compute02:~# service cinder-volume status
cinder-volume start/running, process 10885

root@compute02:~# service quantum-plugin-openvswitch-agent status
quantum-plugin-openvswitch-agent start/running, process 19139
```

You can also use the Nova Client commands to verify the status of Nova services (minus Nova API) across all controller nodes:

```
root@control01:~# nova-manage service list

Binary           Host                                 Zone             Status     State Updated_At
nova-consoleauth control01                            internal         enabled    :-)   2013-09-03
nova-scheduler   control01                            internal         enabled    :-)   2013-09-03
nova-conductor   control01                            internal         enabled    :-)   2013-09-03
nova-cert        control01                            internal         enabled    :-)   2013-09-03
nova-consoleauth control02                            internal         enabled    :-)   2013-09-03
nova-scheduler   control02                            internal         enabled    :-)   2013-09-03
nova-conductor   control02                            internal         enabled    :-)   2013-09-03
nova-cert        control02                            internal         enabled    :-)   2013-09-03
nova-consoleauth control03                            internal         enabled    :-)   2013-09-03
nova-scheduler   control03                            internal         enabled    :-)   2013-09-03
nova-conductor   control03                            internal         enabled    :-)   2013-09-03
nova-cert        control03                            internal         enabled    :-)   2013-09-03
nova-compute     compute02                            nova             enabled    :-)   2013-09-03
nova-compute     compute03                            nova             enabled    :-)   2013-09-03
```

You can also use the Quantum Client commands to verify the status of Quantum Agents across all controller nodes:

```
root@control01:~# quantum agent-list

+--------------------------------------+--------------------+------------------------+-------+----
| id                                   | agent_type         | host                   | alive | adm
+--------------------------------------+--------------------+------------------------+-------+----
| 422e99b2-2780-433d-bf1d-e629db492f8d | Open vSwitch agent | compute03.dmz-pod2.lab | :-)   | Tru
| 66c55a41-d66c-4100-a7ed-ecd2add48100 | Open vSwitch agent | control02.dmz-pod2.lab | :-)   | Tru
| 7ab55d0c-61ef-495f-bb86-2ed6ae549472 | Open vSwitch agent | compute02.dmz-pod2.lab | :-)   | Tru
| 8bd0f911-6a6f-4735-8831-f967a5e792a7 | DHCP agent         | control03.dmz-pod2.lab | :-)   | Tru
| 902947a4-a678-46d3-b0d7-e42060a9c096 | DHCP agent         | control01.dmz-pod2.lab | :-)   | Tru
| 9519c99a-6061-4b72-866c-9b23046e3a20 | Open vSwitch agent | control03.dmz-pod2.lab | :-)   | Tru
| a08605a8-5182-444a-b9d8-bc6dca02cc00 | Open vSwitch agent | control01.dmz-pod2.lab | :-)   | Tru
| c2c2b349-465e-4c04-9285-14005cbba3be | DHCP agent         | control02.dmz-pod2.lab | :-)   | Tru
+--------------------------------------+--------------------+------------------------+-------+----
```

Deploy Your First Instance

Now that the OpenStack HA environment has been deployed and verified, it's now time to spawn Instances.

To spawn Instances, you must first create an image on Glance. Since the HA environment uses Config Drive instead of Nova Metadata, your images must support Cloud Init. Our example deployment uses the Cirros 0.3.1 image.

Copy the Cirros 0.3.1 image to one of the controller nodes:

```
root@control01:~# wget http://download.cirros-cloud.net/0.3.1/cirros-0.3.1-x86_64-disk.img

--2013-09-03 18:28:30--  http://download.cirros-cloud.net/0.3.1/cirros-0.3.1-x86_64-disk.img
Resolving download.cirros-cloud.net (download.cirros-cloud.net)... 69.163.202.251
Connecting to download.cirros-cloud.net (download.cirros-cloud.net)|69.163.202.251|:80... connecte
HTTP request sent, awaiting response... 302 Found
Location: http://cdn.download.cirros-cloud.net/0.3.1/cirros-0.3.1-x86_64-disk.img [following]
--2013-09-03 18:28:30--  http://cdn.download.cirros-cloud.net/0.3.1/cirros-0.3.1-x86_64-disk.img
Resolving cdn.download.cirros-cloud.net (cdn.download.cirros-cloud.net)... 63.80.4.59, 63.80.4.34,
Connecting to cdn.download.cirros-cloud.net (cdn.download.cirros-cloud.net)|63.80.4.59|:80... conn
HTTP request sent, awaiting response... 200 OK
Length: 13147648 (13M) [application/octet-stream]
Saving to: `cirros-0.3.1-x86_64-disk.img'

100%[=========================================================================================>]

2013-09-03 18:28:31 (16.6 MB/s) - `cirros-0.3.1-x86_64-disk.img' saved [13147648/13147648]
```

Create the Cirros image on Glance:

```
root@control01:~# glance image-create --name cirros --is-public=true --disk-format=qcow2 --contain

+------------------+--------------------------------------+
| Property         | Value                                |
+------------------+--------------------------------------+
| checksum         | d972013792949d0d3ba628fbe8685bce     |
| container_format | ovf                                  |
| created_at       | 2013-09-03T18:28:37                  |
| deleted          | False                                |
| deleted_at       | None                                 |
| disk_format      | qcow2                                |
| id               | d90616bd-497b-4a24-86e8-d22dd279079d |
| is_public        | True                                 |
| min_disk         | 0                                    |
| min_ram          | 0                                    |
| name             | cirros                               |
| owner            | 0e65438235af48cca3600bc5b015f72b     |
| protected        | False                                |
| size             | 13147648                             |
| status           | active                               |
| updated_at       | 2013-09-03T18:28:38                  |
+------------------+--------------------------------------+
```

Create your first Quantum tenant network. Our example uses the VLAN ID of 223 for the first tenant network. Keep in mind that the tenant VLAN range is defined within the Puppet site manifest and that your physical network infrastructure must trunk the VLANs within the specified range to all compute and controller nodes:

```
root@control01:~# quantum net-create public223 --tenant_id <tenant-id> --provider:network_type vla
Created a new network:
+---------------------------+--------------------------------------+
| Field                     | Value                                |
+---------------------------+--------------------------------------+
| admin_state_up            | True                                 |
| id                        | fb2bcc59-e338-4b06-90ac-a69d1ba5df3a |
```

```
| name                      | public223                            |
| provider:network_type     | vlan                                 |
| provider:physical_network | physnet1                             |
| provider:segmentation_id  | 223                                  |
| router:external           | False                                |
| shared                    | True                                 |
| status                    | ACTIVE                               |
| subnets                   |                                      |
| tenant_id                 | admin                                |
+---------------------------+--------------------------------------+
```

Create your first Quantum tenant subnet. Our example uses the 192.168.223.x/24 network and 192.168.26.186 for the DNS server. Keep in mind that since the HA architecture uses Quantum Provider Networking Extensions, the gateway for each tenant subnet should be a pair of upstream physcial routers/layer-3 switches configured with a first-hop redundancy protocol such as VRRP:

```
root@control01:~# quantum subnet-create --name 223-subnet --allocation-pool start=192.168.223.10,e
Created a new subnet:
+------------------+--------------------------------------+
| Field            | Value                                |
+------------------+--------------------------------------+
| allocation_pools |                                      |
| cidr             | 192.168.223.0/24                     |
| dns_nameservers  | 192.168.26.186                       |
| enable_dhcp      | True                                 |
| gateway_ip       | 192.168.223.1                        |
| host_routes      |                                      |
| id               | 40c519d6-3cea-40fa-a625-017cef7b189e |
| ip_version       | 4                                    |
| name             | 223-subnet                           |
| network_id       | fb2bcc59-e338-4b06-90ac-a69d1ba5df3a |
| tenant_id        | 0e65438235af48cca3600bc5b015f72b     |
+------------------+--------------------------------------+
```

Create Quantum security group rules to allow the appropriate traffic to your Instances. Our example allows all ICMP and SSH traffic to Instances. Due to a Quantum bug, you may find two default security groups created. You can either delete one of the default groups (before deploying any Instances), or you will need to manage both default groups and use the group ID instead of the group name in your Quantum CLI commands.

To delete one of the default security groups:

```
root@control01:~# quantum security-group-list
+--------------------------------------+---------+-------------+
| id                                   | name    | description |
+--------------------------------------+---------+-------------+
| 12cf3ca2-973c-42a9-9327-78e32d4adafb | default | default     |
| b5840e8d-6019-46c8-8133-f9357eac4ae4 | default | default     |
+--------------------------------------+---------+-------------+

root@control01:~# quantum security-group-delete 12cf3ca2-973c-42a9-9327-78e32d4adafb
Deleted security_group: 12cf3ca2-973c-42a9-9327-78e32d4adafb

root@control01:~# quantum security-group-list
+--------------------------------------+---------+-------------+
| id                                   | name    | description |
+--------------------------------------+---------+-------------+
| b5840e8d-6019-46c8-8133-f9357eac4ae4 | default | default     |
+--------------------------------------+---------+-------------+
```

Create the appropriate security group rules. Again, our example allows all ICMP and SSH traffic to Instances:

```
root@control01:~# quantum security-group-rule-create default --direction ingress --ethertype IPv4

Created a new security_group_rule:
+-------------------+--------------------------------------+
| Field             | Value                                |
+-------------------+--------------------------------------+
| direction         | ingress                              |
| ethertype         | IPv4                                 |
| id                | 447522f7-7184-42e1-ae6d-8cea09eba4d3 |
| port_range_max    |                                      |
| port_range_min    |                                      |
| protocol          | icmp                                 |
| remote_group_id   |                                      |
| remote_ip_prefix  | 0.0.0.0/0                            |
| security_group_id | b5840e8d-6019-46c8-8133-f9357eac4ae4 |
| tenant_id         | admin                                |
+-------------------+--------------------------------------+

root@control01:~# quantum security-group-rule-create default --direction ingress --ethertype IPv4

Created a new security_group_rule:
+-------------------+--------------------------------------+
| Field             | Value                                |
+-------------------+--------------------------------------+
| direction         | ingress                              |
| ethertype         | IPv4                                 |
| id                | 8b5eccb6-73a8-47ba-8940-08be0e2bc483 |
| port_range_max    | 22                                   |
| port_range_min    | 22                                   |
| protocol          | tcp                                  |
| remote_group_id   |                                      |
| remote_ip_prefix  | 0.0.0.0/0                            |
| security_group_id | b5840e8d-6019-46c8-8133-f9357eac4ae4 |
| tenant_id         | admin                                |
+-------------------+--------------------------------------+
```

Now that an image is created and a tenant network, subnet and security rules have been created, you should be able to boot an Instance. Our example uses the tiny flavor. You can obtain the net-id from the quantum net-list command and c1 is the name of our Instance:

```
root@control01:~# nova boot --image cirros --flavor m1.tiny --nic net-id=fb2bcc59-e338-4b06-90ac-a

+-------------------------------------+--------------------------------------+
| Property                            | Value                                |
+-------------------------------------+--------------------------------------+
| status                              | BUILD                                |
| updated                             | 2013-09-03T18:52:11Z                 |
| OS-EXT-STS:task_state               | scheduling                           |
| OS-EXT-SRV-ATTR:host                | None                                 |
| key_name                            | None                                 |
| image                               | cirros                               |
| hostId                              |                                      |
| OS-EXT-STS:vm_state                 | building                             |
| OS-EXT-SRV-ATTR:instance_name       | instance-00000001                    |
| OS-EXT-SRV-ATTR:hypervisor_hostname | None                                 |
| flavor                              | m1.tiny                              |
| id                                  | 2523e675-da7b-4bc1-893a-8cab44074d1d |
| security_groups                     | [{u'name': u'default'}]              |
| user_id                             | c921e800213141618610e57c1c8d4873     |
```

```
| name                                 | c1                                   |
| adminPass                            | WJtKrhmjQ48P                         |
| tenant_id                            | 0e65438235af48cca3600bc5b015f72b     |
| created                              | 2013-09-03T18:52:11Z                 |
| OS-DCF:diskConfig                    | MANUAL                               |
| metadata                             | {}                                   |
| accessIPv4                           |                                      |
| accessIPv6                           |                                      |
| progress                             | 0                                    |
| OS-EXT-STS:power_state               | 0                                    |
| OS-EXT-AZ:availability_zone          | nova                                 |
| config_drive                         |                                      |
+--------------------------------------+--------------------------------------+
```

It may take a minute or two for the Instance to spawn. You can use the Nova list or show command to verify the status:

```
root@control01:~# nova list

+--------------------------------------+------+--------+------------+-------------+----------------
| ID                                   | Name | Status | Task State | Power State | Networks
+--------------------------------------+------+--------+------------+-------------+----------------
| 2523e675-da7b-4bc1-893a-8cab44074d1d | c1   | ACTIVE | None       | Running     | public223=192.
+--------------------------------------+------+--------+------------+-------------+----------------

root@control01:~# nova show c1

+--------------------------------------+--------------------------------------------------------------+
| Property                             | Value                                                        |
+--------------------------------------+--------------------------------------------------------------+
| status                               | ACTIVE                                                       |
| updated                              | 2013-09-03T18:52:21Z                                         |
| OS-EXT-STS:task_state                | None                                                         |
| OS-EXT-SRV-ATTR:host                 | compute03                                                    |
| key_name                             | None                                                         |
| image                                | cirros (d90616bd-497b-4a24-86e8-d22dd279079d)                |
| hostId                               | 92123d378542a9e6fc1553d4dd4aa05da2b7595d7f99409659477578 |
| OS-EXT-STS:vm_state                  | active                                                       |
| OS-EXT-SRV-ATTR:instance_name        | instance-00000001                                            |
| OS-EXT-SRV-ATTR:hypervisor_hostname | compute03.dmz-pod2.lab                                       |
| flavor                               | m1.tiny (1)                                                  |
| id                                   | 2523e675-da7b-4bc1-893a-8cab44074d1d                         |
| security_groups                      | [{u'name': u'default'}]                                      |
| public223 network                    | 192.168.223.10                                               |
| user_id                              | c921e800213141618610e57c1c8d4873                             |
| name                                 | c1                                                           |
| created                              | 2013-09-03T18:52:11Z                                         |
| tenant_id                            | 0e65438235af48cca3600bc5b015f72b                             |
| OS-DCF:diskConfig                    | MANUAL                                                       |
| metadata                             | {}                                                           |
| accessIPv4                           |                                                              |
| accessIPv6                           |                                                              |
| progress                             | 0                                                            |
| OS-EXT-STS:power_state               | 1                                                            |
| OS-EXT-AZ:availability_zone          | nova                                                         |
| config_drive                         |                                                              |
+--------------------------------------+--------------------------------------------------------------+
```

Make not of the Instance's IP address. From a host outside of the OpenStack environment and with access to the tenant network (192.168.223.x/24 in our example), you should be able to SSH to the Instance. cirros/cubswin:) are the login credentials for Cirros-based Instances:

```
ssh cirros@192.168.223.10

The authenticity of host '192.168.223.10 (192.168.223.10)' can't be established.
RSA key fingerprint is 89:93:e7:a5:26:ee:b3:5b:cb:e6:3d:9f:8f:b5:58:2f.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.223.10' (RSA) to the list of known hosts.
cirros@192.168.223.10's password:
$ ifconfig
eth0      Link encap:Ethernet  HWaddr FA:16:3E:05:3F:31
          inet addr:192.168.223.10  Bcast:192.168.223.255  Mask:255.255.255.0
          inet6 addr: fe80::f816:3eff:fe05:3f31/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:70 errors:0 dropped:0 overruns:0 frame:0
          TX packets:49 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:9092 (8.8 KiB)  TX bytes:7377 (7.2 KiB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

You can verify connenctivity from the Instance to its default gateway (upstream physical router pair):

```
$ ping 192.168.223.1
PING 192.168.223.1 (192.168.223.1): 56 data bytes
64 bytes from 192.168.223.1: seq=0 ttl=255 time=1.178 ms
64 bytes from 192.168.223.1: seq=1 ttl=255 time=1.386 ms
64 bytes from 192.168.223.1: seq=2 ttl=255 time=1.366 ms
^C
--- 192.168.223.1 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 1.178/1.310/1.386 ms
```

If your DNS is properly configured (outside the scop of this document), you should also be able to verify connectivity using DNS names:

```
$ ping www.google.com
PING www.google.com (74.125.129.106): 56 data bytes
64 bytes from 74.125.129.106: seq=0 ttl=41 time=24.175 ms
64 bytes from 74.125.129.106: seq=1 ttl=41 time=24.432 ms
64 bytes from 74.125.129.106: seq=2 ttl=41 time=24.158 ms
```

Deploy Your First Volume

Now that the OpenStack HA environment has been deployed and you have successfully spawned an Instance, it's time to create and attach a Cinder volume.

Create a Cinder volume. Our example uses the v1 for the volume name with a 1GB size:

```
root@control01:~# cinder create --display-name v1 1
+---------------------+--------------------------------------+
|       Property      |                Value                 |
+---------------------+--------------------------------------+
|     attachments     |                  []                  |
|  availability_zone  |                 nova                 |
```

```
|       bootable      |                false                |
|      created_at     |      2013-09-03T19:02:45.266569      |
| display_description |                None                 |
|     display_name    |                 v1                  |
|          id         | b01f4a98-866b-4e5e-8e36-9052b3fe9e40 |
|       metadata      |                 {}                  |
|         size        |                 1                   |
|     snapshot_id     |                None                 |
|     source_volid    |                None                 |
|        status       |              creating               |
|     volume_type     |                None                 |
+---------------------+-------------------------------------+
```

Attach the volume to an instance:

```
nova volume-attach <instance_name> <volume_id> /dev/vdc
```

Example:

```
root@control01:~# nova volume-attach c1 b01f4a98-866b-4e5e-8e36-9052b3fe9e40 /dev/vdc
+----------+--------------------------------------+
| Property | Value                                |
+----------+--------------------------------------+
| device   | /dev/vdc                             |
| serverId | 2523e675-da7b-4bc1-893a-8cab44074d1d |
| id       | b01f4a98-866b-4e5e-8e36-9052b3fe9e40 |
| volumeId | b01f4a98-866b-4e5e-8e36-9052b3fe9e40 |
+----------+--------------------------------------+
```

Verify the status of the volume attachment:

```
root@control01:~# nova volume-list
+--------------------------------------+--------+--------------+------+-------------+-------------
| ID                                   | Status | Display Name | Size | Volume Type | Attached to
+--------------------------------------+--------+--------------+------+-------------+-------------
| b01f4a98-866b-4e5e-8e36-9052b3fe9e40 | in-use | v1           | 1    | None        | 2523e675-da7
+--------------------------------------+--------+--------------+------+-------------+-------------
```

Configure Quantum DHCP Agent Redundancy

By default, Quantum deploys only one DHCP server per network. To provide DHCP server redundancy for Quantum tenant networks, follows the steps in the Manual Deployment Guide.

Support

Email: openstack-support@cisco.com

Credits

This work has been based on:

- OpenStack Grizzly Installation Guide for Ubuntu 12.04 (LTS) Documentation
- Cisco High-Availability Manual Installation Guide

Authors

Daneyon Hansen danehans@cisco.com