Reference Architecture for OpenStack Grizzly

OpenStack is one of the fastest growing open source projects today, with thousands of active developers and hundreds of actively supporting companies and individuals. Getting started with OpenStack has been simplified by the existence of this large development community to such a degree that a single script can turn a virtual machine into a usable OpenStack-based test environment. But in attempting to make a production-ready system, the choices suddenly narrow and deploying a system that can be managed after production starts is even more difficult.

The Cisco Reference Architecture for OpenStack Grizzly is one of the current models for getting to that manageable deployment. It includes a model for the compute, network, and storage components, the virtualization platform, and also how to install the system. A set of guidelines and a validated process for using well-known Hypervisor and OpenStack code distributions, along with a tested and validated DevOps deployment model based on the work of the community (driven by Puppet Labs with their Puppet DevOps toolset), is included. This document will also cover the tests that have been completed against the environment to highlight where the system is functional, and in some cases, areas of possible concern where improvements will be made.

Reference Architecture

The reference architecture supports a common model for OpenStack deployments, meeting a typical set of user and usage requirements. Aspects of the system can be modified and scaled either up or down, depending on specific end user needs, but the system is based on a few key tenets:

1. Middle of the system specs to meet "average" scenarios.
2. Provide for future extension to support a high availability service (not covered in this document).
3. Minimize system differences to reduce Day0 (initial) and Day 2 (break/fix) operational requirements.
4. Avoid top bin devices/components to manage overall system cost and improve system value.
5. Cover the OpenStack core components.
6. Enhance the core components when appropriate (for services such as future High Availability control plane or added functionality such as Service Assurance tools).

Reference Architecture Target Requirements

In addition to the general model requirements, we will also address a set of feature/capability requirements in future testing iterations using Cisco Reference Architecture for OpenStack Grizzly. Namely:

- Support for micro, small, medium, large, x-large, and xx-large instances (up to 32GB memory and 8 cores per VM).
- Support for local instance persistence.
- Ability to support VM migration and restart.
- Support for single interface or multi-interface networks.
- Support OpenStack Essex, Folsom, and Grizzly releases against Ubuntu 12.04 LTS and RedHat RHEL 6.4.
- Future support for automated deployment of our highly available access to OpenStack services configuration (Nova, Glance, Keystone, Quantum, Swift, and Cinder).

Physical Infrastructure Model

To simplify operational management, only two types of systems are included in the model: compute-centric (Table 1) and storage-centric (Table 2).

Table 1 Compute Model based on UCS C220-M3

Element	Type	Quantity	Description
CPU	Intel E5-2660	2	Mid-tier high core count CPU for a balance of power and VM scale.
Memory	1600MHz 16GB dual rank DIMM	16	Supports up to 4 xx-large instances per physical system.
NIC	Cisco VIC	1	Provides dual port 10G interfaces for resiliency and the ability to support VM-FEX for hypervisor bypass (on supported Hypervisors).
Disk Controller	Mega-RAID 9266i	1	Provide memory-backed RAID operation across the local disks, improve performance of lower cost/denser disk options. RAID 10 for performance
Disk Drives	600GB 10Krpm SAS	8	Provide a large possible footprint for local VM instances with reasonable performance.

The compute system is based on the 1RU C220-M3 platform and leverages a low power 8 core CPU and 256GB of memory giving a memory-to-core ratio of 16:1. The storage subsystem is based on a high performance RAID controller and 8 SAS disks for a flexible model for distributed CINDER and/or Ceph storage. The network interface is based on the Cisco Virtual Interface Controller (VIC), providing dual 10Gbps network channels and enabling Hypervisor Bypass with Virtual Machine Fabric Extension (VM-FEX) functionality when combined with a Nexus 5500 series data center switch as the Top of Rack (TOR) device, Fibre Channel over Ethernet (FCOE) storage, and Network Interface Card (NIC) bonding for network path resiliency or increased network performance for video streaming, high performance data moves, or storage applications.

Table 2 Storage Model based on UCS C240-M3

Element	Type	Quantity	Description
CPU	Intel E5-2609	2	Lower core count CPU for a reduced computational non-complex workload.
Memory	1600MHz 8GB dual rank DIMM	4	Provides working memory for in-system disk cache.
NIC	Cisco VIC	1	Provides dual port 10G interfaces for bonded NIC resiliency.
Disk Controller	Mega-RAID 9266i	1	Provide memory-backed RAID operation across the local disks for non-Swift-based storage. No RAID config.
Disk Drives	1 TB 7.2Krpm SATA-6	24	Disk for Swift or block or NAS depending on usage model.

The storage system is based on the 2RU C240-M3 platform, which is similar at the baseboard level to the C220-M3, but provides up to 24 2.5" drive slots. With 24 spindles, this platform is focused on storage as opposed to compute, and while it could be used as configured for a combined all-in-one platform, the reference makes use of dual low power 4 core CPUs, and a much smaller memory space at 32GB total, which is our current model for managing SWIFT or CINDER-focused nodes specifically. This platform also includes the Cisco VIC for up to 20Gbps of storage forwarding with link resiliency when combined with the dual TOR model.

Compute BIOS

The current default host BIOS configuration is appropriate for deployment; however, it is convenient to change some of the parameters to accelerate boot, and address hardware differences (such as having the Cisco FlexFlash installed). This will improve the automated deployment process. The manual steps required to implement these changes can be found in the Cisco UCS C-Series Servers Integrated Management Controller CLI Configuration Guide or the Cisco UCS C-Series Servers Integrated Management Controller Configuration Guide for CLI or Web UI based configuration.

Some of the non-standard recommended parameters are defined in Table 3:

Table 3 Non-Standard Compute BIOS Recommended Parameters

BIOS Parameter	Value	Description
bios/LomOpromControlPort0	Disabled	Disable un-used LOM port BIOS. There are either 2 or 4 of these (0,1 or 0,1,2,3) for C220 vs. C240.
bios/UsbPortInt	Disabled	Access to the internal 4GB USB card if installed (not in the reference model).
bios/UsbPortSdCard	Disabled	Access to the internal 16GB Cisco FlexFlash if installed (not in the reference model).
Boot-order	cdrom,pxe,hdd	Simplifies day 0 OS install, PXE being the principal method for installing the hypervisor OS.
BIOS/CIMC Version	1.5(f)	The current 1.5 BIOS release provides enhanced CIMC-based access to BIOS and PCI device management.

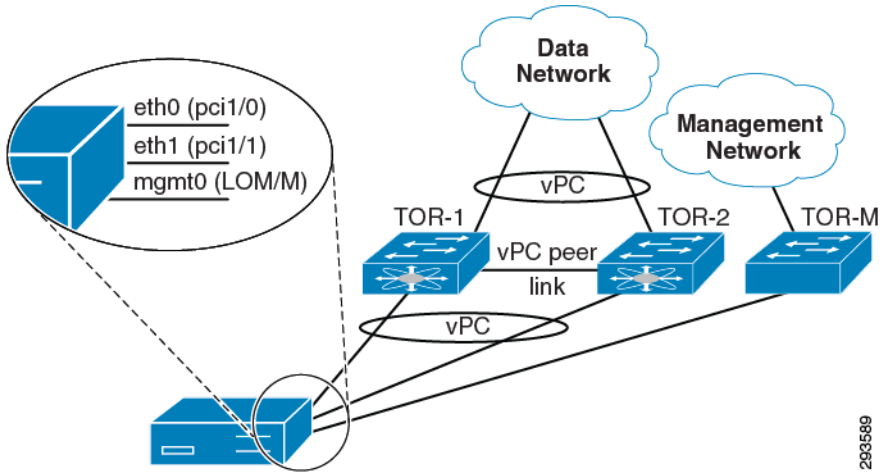
A set of utility scripts are available to facilitate BIOS updates and storage configurations, along with collecting data needed for the reference deployment model. These scripts are available from the Cisco Systems GitHub repository:

https://github.com/CiscoSystems/ucs-openstack-cimc-expect.git

Network Model

The upstream network is based on the Nexus 5500 series switch enabling the use of a number of advanced scale-out network services in the Layer 2 (Link Local) and Layer 3 (Routed Network) services. In the basic reference, the TOR switches are configured as a virtual Port Channel (vPC) pair, with a set of 10Gigabit connections between them as the VPC peer link, and a set of two ports each as the vPC to the upstream device (a Nexus 7000 series device in our Virtual Multiservice Datacenter (VMDC) network model). Figure 1 shows the normal link configurations, and Table A-1 in Appendix A shows an example port mapping for the two TOR switches with the server management interfaces (TOR-M in Figure 1) collapsed onto the TOR-1 and TOR-2 devices.

Figure 1 Physical Network Configuration



The physical model above makes the following assumptions:

- A separate interface and VLAN is used for management, both CIMC (power, KVM, and bare metal management) and for the hypervisor management interface.
- An additional VLAN (or in certain network models, additional VLANs) will be provisioned on the trunk port(s) on a separate interface associated with the hypervisor, by the Cisco Quantum plugin.
- vPC is used to connect to an upstream router (or routers) who support LACP port channels at a minimum.
- vPC is used to connect to the physical servers, but this is either manually or automatically configured after the base OS is installed due to the inability of most Linux installers to PXE boot and install over a bonded network interface.

Logically, the network is segregated either via VLANs (as proposed in this reference) or via an overlay L2 technology like VXLAN. The latter is expected to become the standard mechanism for providing much greater tenancy or project-level L2 segregation scale by the Fall 2013 OpenStack release time frame (the Havana release). L3 and other network services have only limited support in the current Folsom and Grizzly releases, and in our reference architecture, the L3 services will be supported by the L3_Agent network model. Security will be provided by the IPtables security and NAT functionality driven by Quantum.

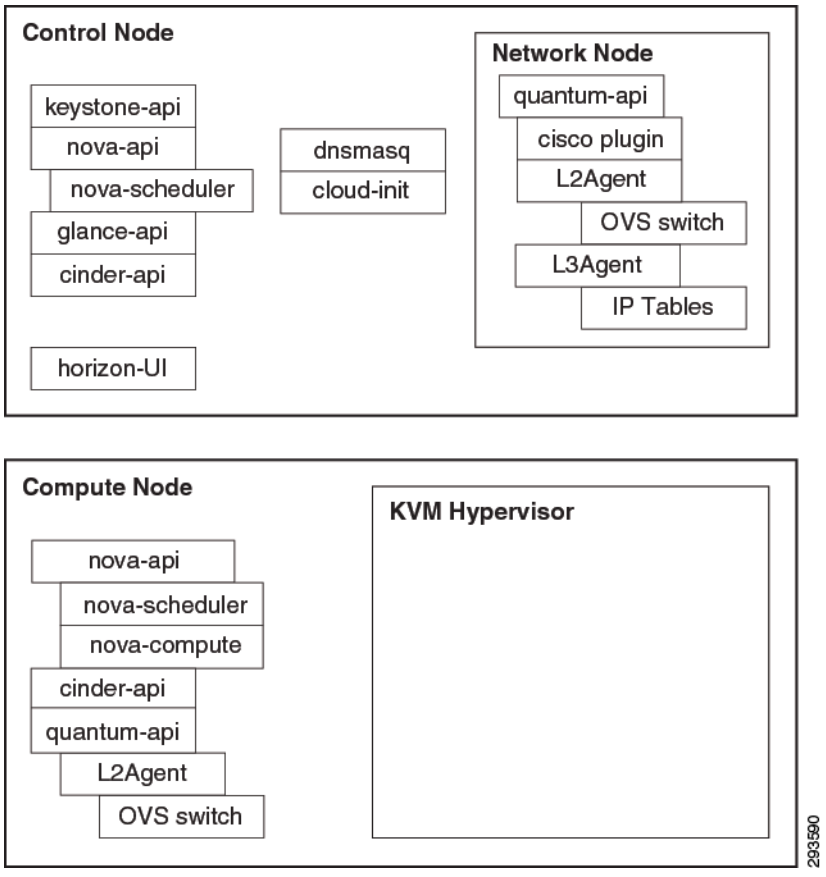
The vPC enabled TOR switches we recommend are either the Nexus 3548 switch with L3 services if performance is the principal concern, or the Nexus 5548-UP with the L3 daughter card if features like VM-FEX and/or FCOE capability are of interest. Nexus 3k is a low-latency HPC switch. Nexus 5k is a popular access switch and is included in Flexpod, vBlock and VMDC. A BOM for both devices is included in Appendix B: Bills of Material, page -17.

Software Architecture

The system software architecture for the Grizzly release of Cisco OpenStack Installer on Ubuntu 12.04 LTS is straightforward (Figure 2). Future references will be released to support the Havana release, and RedHat-based Hypervisor OS systems.

The non-HA model has a single node acting as a control node, running the control APIs, with the exception of nova-compute. This same system also runs the Quantum L2 and L3 agents providing local to external network connectivity and security.

Figure 2 Primary Software Components on Control and Compute Nodes



The current deployment model also includes a build node, which provides a number of additional services beyond the OpenStack-specific deployment components highlighted above, namely:

- Cobbler (https://github.com/cobbler) to support bare metal install of the compute node hypervisor and control node base OS.

- Puppet Agent and Puppet Master (https://puppetlabs.com) DevOps toolset for system management and deployment services.
- Nagios, Collectd, and Graphite for system health data collection from the control and compute nodes.

Small Scale Example System

As most deployments don't immediately jump to a full rack or even multi-rack OpenStack systems deployment, we have provided a small system model that should allow a simple update to a complete rack model. This simple model starts with a single switch and a set of three compute class nodes.

The following system is an example of a small test or application support deployment setup model. This system is designed to support ~48 virtual compute nodes across this particular environment and to be configured into a single 42 RU server rack with the intention of having additional devices added to build out a full server rack-based system (Table 4).

Table 4 Small Scale System Rack Layout

Location in Rack (RU number)	Principal Function	Component Name	Element
Slot 42 (top of rack)	Network	TOR-1	5548-UP
Slot 41-39	Expansion		Blank
Slot 38	Control/Network	build-server	C220-M3 Compute
Slot 37	Control/Network	control-server	C220-M3 Compute
Slot 35-36	Expansion		Blank
Slot 34	Compute	build-server 01	C220-M3 Compute
Slot 33	Compute	build-server 02	C220-M3 Compute

The physical network model for this system is also quite simple, but is designed to be ready to scale out to the full rack system as well and uses the same port model, but without any of the redundant links installed initially (and no virtual port channel or redundant uplinks configured). Refer to Appendix A: Switch Port Mapping, page -15 for the wiring diagram.

Rack Scale Example System

This is a system that includes all of the components (Table 5) needed for the upcoming HA systems model, along with the SWIFT and CINDER storage components.

Table 5 Rack Scale Example System

Location in Rack (RU number)	Principal Function	Component Name	Element
Slot 42 (top of rack)	Network	TOR-1	5548-UP
Slot 41	Network	TOR-2	5548-UP
Slot 40			Expansion
Slot 39			Expansion
Slot 38	Build	build-server	C220-M3 Compute

Table 5 Rack Scale Example System (continued)

Slot 37	Control/Compute	control-server01	C220-M3 Compute
Slot 36	Control/Compute	control-server02	C220-M3 Compute
Slot 35	Control/Compute	control-server03	C220-M3 Compute
Slot 34	Compute	compute-server01	C220-M3 Compute
Slot 33	Compute	compute-server02	C220-M3 Compute
Slot 32	Compute	compute-server03	C220-M3 Compute
Slot 31	Compute	compute-server04	C220-M3 Compute
Slot 30	Compute	compute-server05	C220-M3 Compute
Slot 29	Compute	compute-server06	C220-M3 Compute
Slot 28	Compute	compute-server07	C220-M3 Compute
Slot 27	Compute	compute-server08	C220-M3 Compute
Slot 26	Compute	compute-server09	C220-M3 Compute
Slot 25	Compute	compute-server10	C220-M3 Compute
Slot 24	Compute	compute-server11	C220-M3 Compute
Slot 23	Compute	compute-server12	C220-M3 Compute
Slot 22	Compute	compute-server13	C220-M3 Compute
Slot 21	Compute	compute-server14	C220-M3 Compute
Slot 20	Compute	compute-server15	C220-M3 Compute
Slot 19	Storage Proxy	proxy-server01	C220-M3 Compute
Slot 18	Storage Proxy	proxy-server02	C220-M3 Compute
Slot 17	Storage Proxy	proxy-server03	C220-M3 Compute
Slot 15-16	Cinder Block	block-server01	C240-M3 Compute
Slot 13-14	Cinder Block	block-server02	C240-M3 Compute
Slot 11-12	Swift	swift-server01	C240-M3 Compute
Slot 9-10	Swift	swift-server02	C240-M3 Compute
Slot 7-8	Swift	swift-server03	C240-M3 Compute
Slot 5-6	Swift	swift-server04	C240-M3 Compute
Slot 3-4	Swift	swift-server05	C240-M3 Compute
Slot 1-2			Expansion

Systems Installation

The following section walks through the software steps required to install RedHat RDO on top of the Cisco Reference Architecture for OpenStack system. This process presumes an environment as described above.

Assumptions

Although other configurations are supported, the following instructions target an environment with a build node, a controller node, and at least one compute node. Additional compute nodes may optionally be added.

Also, these instructions primarily target deployment of OpenStack onto UCS servers (either blades or rack-mount form factors). Several steps in the automation leverage the UCS manager or CIMC to execute system tasks. Deployment on non-UCS gear may well work, particularly if the gear has functional IPMI, but may require additional configuration or additional manual steps to manage systems.

Cisco OpenStack Installer Grizzly requires that you have two physically or logically (VLAN) separated IP networks. One network is used to provide connectivity for OpenStack API endpoints, Open vSwitch (OVS) GRE endpoints and OpenStack/UCS management. The second network is used by OVS as the physical bridge interface and by Quantum as the public network.

Creating a Build Server

To create a build server, perform the following:

Step 1 To deploy Cisco OpenStack, first configure a build server.

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 eventual management interfaces of the OpenStack cluster machines are the minimal requirements. This machine can be physical or virtual; eventually a pre-built VM of this server will be provided, but this is not yet available.

Step 2 Install Ubuntu 12.04 LTS onto this build server.

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.

- When the installation finishes, log in and become root: sudo -H bash

Note If you have proxies, or your control and compute nodes do not have Internet access, please read the following:

- 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.
- You have two choices for setting up the build server. 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.

Model 1: Run the Script

Step 3 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 |
/bin/bash
```

With a proxy, use:

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

Step 4 You can now jump to Customizing the Build Server. Otherwise, follow the steps below.

Model 2: Run the Commands Manually

Step 5 Install any pending security updates:

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

Note The system may need to be restarted after applying the updates.

Get the reference example manifests. Under the grizzly-manifests GitHub repository, you will find different branches, so select the one that matches your topology plans most closely. In the following examples, the multi-node branch will be used, which is likely the most common topology:

```
git clone https://github.com/CiscoSystems/grizzly-manifests ~/cisco-grizzly-manifests/
cd ~/cisco-grizzly-manifests
git checkout -q g.0
```

With a proxy:

```
https_proxy=http://proxy.example.com:80 git clone
https://github.com/CiscoSystems/grizzly-manifests ~/cisco-grizzly-manifests/
cd ~/cisco-grizzly-manifests
https_proxy=http://proxy.example.com:80 git checkout -q g.0
```

Step 6 Copy the Puppet manifests from ~/cisco-grizzly-manifests/manifests/ to /etc/puppet/manifests/

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

Step 7 Copy the Puppet templates from ~/cisco-grizzly-manifests/templates/ to /etc/puppet/templates/

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

Step 8 Then get the reference Puppet modules from Cisco's GitHub repository:

```
(cd /etc/puppet/manifests; sh /etc/puppet/manifests/puppet-modules.sh)
```

With a proxy:

```
(cd /etc/puppet/manifests; http_proxy=http://proxy.example.com:80\
https_proxy=http://proxy.example.com:80 sh\ /etc/puppet/manifests/puppet-modules.sh)
```

Customizing the Build Server

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

```
clean_node.sh
cobbler-node.pp
core.pp
modules.list
puppet-modules.sh
reset_nodes.sh
site.pp.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 user-modifiable components and defines the various parameters that must be set to configure the OpenStack cluster, including the Puppet Master and Cobbler setup on the build server. 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.

Step 1 IMPORTANT! You must copy site.pp.example to site.pp and then edit it as appropriate for your installation. It is internally documented.

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

Step 2 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 client on the build server will follow the instructions in the site.pp and cobbler-node.pp manifests and will configure several programs on the build server:

- Network Time Protocol daemon (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**: an 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 server will take several minutes to complete as it downloads, installs, and configures all the software needed for these applications.

Step 3 Once the site.pp 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
```

Step 4 After the build server is configured, the systems listed in site.pp should be defined in cobbler on the build server:

```
cobbler system list
control-server
compute-server01
compute-server02
```

Deploying OpenStack

Perform the following steps to deploy the control, compute, and storage nodes.

Step 1 Use cobbler to build your controller:

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

Replace node_name with the name of your controller.

```
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.

Step 2 You can watch the progress on the console of your controller node as Cobbler completes the automated install of Ubuntu. Once the installation finishes, the controller node will reboot and then will run Puppet after it boots up. Puppet will pull and apply the controller node configuration defined in the Puppet manifests on the build server.

This step will take several minutes, as Puppet downloads, installs, and configures the various OpenStack components and support applications needed on the control node. /var/log/syslog on the controller node will display the progress of the Puppet configuration run.

Note It may take more than one Puppet run for the controller node to be set up completely, especially if there are proxies in the path as some proxies can have issues with apt-get installs and updates. Observe the log files (/var/log/syslog on the controller node) to verify that the controller configuration has converged completely to the configuration defined in Puppet.

Step 3 Once the Puppet configuration of the controller has completed, follow the same steps to build each of the other nodes in the cluster, using clean_node.sh to initiate each install. As with the controller, the other nodes will take several minutes for Puppet configuration to complete and may require multiple runs of Puppet before they are fully converged to their defined configuration state.

As a short cut, if you want to build all of the nodes defined in your cobbler-node.pp file, you can run:

```
for n in `cobbler system list`; do clean_node.sh $n ; done
```

Or you can run a full reset script, which also does this and re-runs the build-node Puppet apply and Puppet plugin download steps:

```
./reset_nodes.sh
```

Step 4 Bring up the storage nodes by running "cobbler system poweron --name=[storage node name]". Allow the operating system to be installed and for the puppet agent to complete it's first catalog run.

Step 5 Bring up the Swift proxy node by running "cobbler system poweron --name=[proxy node name]". Allow the operating system to be installed and for the puppet agent to complete it's first catalog run.

Step 6 Allow puppet to make another catalog run on each storage node.

Step 7 Once the OpenStack nodes have been built using Cobbler, 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.

Testing OpenStack

Once the nodes are built, and once Puppet runs have completed on all nodes (watch /var/log/syslog on the cobbler node), you should be able to log into the OpenStack Horizon interface:

```
http://ip-of-your-control-node/ user: admin, password: Cisco123 (if you didn't change
the defaults in the site.pp file).
```

You will still need to log into the console of the control node to load in an image using user: localadmin, password: ubuntu.

If you SU to root, you will need to source the openrc auth file, which is in the root's home directory (run "source openrc" in /root/), and you can launch a test file in /tmp/nova_test.sh.

Deploying Your First VM

The following deployment steps should be used after completing clean puppet runs on OpenStack Nodes and restarting quantum-server and quantum-plugin-openvswitch-agent services.

Manual Process

Step 1 Create quantum public network.

```
quantum net-create public --router:external=True
```

Step 2 We are using 192.168.221.0/24 as our external network.

Note The eth settings on the Controller Node associated to this network should not have an IP address assigned to it as it will function in bridged mode.

```
quantum subnet-create public 192.168.221.0/24
```

Note If there are upstream routers/L3 switches that use HSRP/GLBP/VRRP that use low-order IP addresses such as .2 and .3 then the default subnet address assignments used by Quantum for this subnet (such as floating IP addresses and the Qrouter interface [default is .3]) will directly conflict with these real IP addresses on the upstream first hop routers. You can alter these default address assignments for the Quantum subnet by using the "--allocation-pool" range when creating the Quantum subnet. The example that follows will use the default upstream router address of .1 (in this example the upstream HSRP address would be 192.168.221.1) and the first addresses for floating-IPs will begin at .10:

```
quantum subnet-create --tenant-id services --allocation-pool
start=192.168.221.10,end=192.168.221.250 public 192.168.221.0/24
```

Step 3 Create the internal (data) network used for Tenants. Create additional networks and associated subnets as needed. In the example below, we are assigning specific DNS servers that will be used by the instances.

```
quantum net-create net10
quantum subnet-create net10 10.10.10.0/24 --name net10-subnet --dns_nameservers
list=true 8.8.8.8 8.8.4.4
```

Note Replace 8.8.8.8 8.8.4.4 with the IP address(es) of the DNS server or servers virtual machines should use.

Step 4 Create a virtual router and an associated interface used for the subnet created in the previous step:

```
quantum router-create router1
quantum router-interface-add router1 net10-subnet
```

Step 5 Connect the virtual router to your external network:

```
quantum router-gateway-set router1 public
```

Step 6 Download an image and add it to Glance:

a. For Ubuntu Precise:

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

glance add name="precise-x86_64" is_public=true container_format=ovf disk_format=qcow2
< precise-server-cloudimg-amd64-disk1.img
```

b. For Cirros Cloud Image:

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

glance add name="cirros-x86_64" is_public=true disk_format=qcow2 container_format=ovf
< cirros-0.3.1-x86_64-disk.img
```

Step 7 Create an SSH keypair and add the public key to Nova. Make sure you create a key-pair for your Network and Controller Nodes. Note: leave the passphrase empty when creating the keypair:

```
ssh-keygen
```

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

Step 8 Boot an Instance (Precise image example):

```
quantum net-list
nova boot --image precise-x86_64 --flavor m1.tiny --key_name <key_name> --nic
net-id=<quantum-net10-id> <your_instance_name>
```

a. Cirros Image Example

```
nova boot --image cirros-x86_64 --flavor m1.tiny --key_name <key_name> --nic
net-id=<quantum-net10-id> <your_instance_name>
```

b. Verify that your instance has spawned successfully:

```
nova show <your_instance_name>
```

Step 9 Verify connectivity to Instance from the node running Quantum L3 Agent (Controller Node). Since we are using namespaces, we run the commands from the context of the qrouter using the "ip netns exec qrouter" syntax. Below, we list the qrouter to get its router-id, we connect to the qrouter and get a list of its addresses, we ping the instance from the qrouter and then we SSH into the instance from the qrouter:

```
quantum router-list
ip netns exec qrouter-<quantum-router-id> ip addr list
ip netns exec qrouter-<quantum-router-id> ping <fixed-ip-of-instance>
ip netns exec qrouter-<quantum-router-id> ssh ubuntu@<fixed-ip-of-instance>
```

Note You can get the internal fixed IP of your instance with the following command: nova show <your_instance_name>

Step 10 Create and associate a Floating IP. You will need to get a list of the networks copy the correct IDs:

```
quantum net-list
quantum floatingip-create <public/ext-net-id>
quantum floatingip-associate <floatingip-id> <internal VM port-id>
or in a single step:
quantum net-list
quantum port-list
quantum floatingip-create --port_id <internal VM port-id> <public/ext-net-id>
```

Step 11 Enable Ping and SSH to Instances:

```
nova secgroup-add-rule default icmp -1 -1 0.0.0.0/0
nova secgroup-add-rule default tcp 22 22 0.0.0.0/0
```

Step 12 Ping and SSH to your Instances from an external host.

Running Your OpenStack Environment

If the previous set of instructions was followed correctly, you should now have a simple system image, security and network access, and manipulate and map storage to running instances. Future OpenStack white papers will look at managing users and projects, scaling systems, running multi-site systems, and a host of other operations and scale out tasks. Check back often for more from the OpenStack team.

Appendix A: Switch Port Mapping

Table 6 provides an example of switch port mapping.

Table 6 Example Switch Port Mapping

Switch Port	Trunk/Access	VLANs	Speed	Connected Device	Device Port	Port-Channel	vPC
		native, allowed					
e0/1	trunk	100,all	10G	Upstream	PC-100 p1	98	98
e0/2	trunk	100,all	10G	Upstream	PC-100 p2	98	98
e0/3	trunk	100,all	10G	TOR-2	e0/3	99	99
e0/4	trunk	100,all	10G	TOR-2	e0/4	99	99
e0/5	trunk	100,100	10G	compute-00	pci1/0	100	100
e0/6	trunk	100,100	10G	compute-01	pci1/0	101	101
e0/7	trunk	100,100	10G	compute-02	pci1/0	102	102
e0/8	trunk	100,100	10G	compute-03	pci1/0	103	03
e0/9	trunk	100,100	10G	compute-04	pci1/0	104	104
e0/10	trunk	100,100	10G	compute-05	pci1/0	105	105
e0/11	trunk	100,100	10G	compute-06	pci1/0	106	106
e0/12	trunk	100,100	10G	compute-07	pci1/0	107	107
e0/13	trunk	100,100	10G	compute-08	pci1/0	108	108
e0/14	trunk	100,100	10G	compute-09	pci1/0	109	109
e0/15	trunk	100,100	10G	compute-10	pci1/0	110	110
e0/16	trunk	100,100	10G	compute-11	pci1/0	111	111
e0/17	trunk	100,100	10G	compute-12	pci1/0	112	112
e0/18	trunk	100,100	10G	compute-13	pci1/0	113	113
e0/19	trunk	100,100	10G	compute-14	pci1/0	114	114
e0/20	trunk	100,100	10G	compute-15	pci1/0	115	115
e0/21	trunk	100,100	10G	compute-16	pci1/0	116	116
e0/22	trunk	100,100	10G	compute-17	pci1/0	117	117
e0/23	trunk	100,100	10G	compute-18	pci1/0	118	118
e0/24	trunk	100,100	10G	compute-19	pci1/0	119	119
e0/25	trunk	100,100	10G	storage-00	pci1/0	120	120
e0/26	trunk	100,100	10G	storage-01	pci1/0	121	121
e0/27	trunk	100,100	10G	storage-02	pci1/0	122	122
e0/28	trunk	100,100	10G	storage-03	pci1/0	123	123
e0/29	trunk	100,100	10G	storage-04	pci1/0	124	124
e0/30	trunk	100,100	10G	storage-05	pci1/0	125	125
e0/31	trunk	100,100	10G	storage-06	pci1/0	126	126
e0/32	trunk	100	1G	compute-00	lot-m		

Table 6 Example Switch Port Mapping (continued)

Switch Port	Trunk/Access	VLANs	Speed	Connected Device	Device Port	Port-Channel	vPC
e0/33	access	100	1G	compute-02	lot-m		
e0/34	access	100	1G	compute-04	lot-m		
e0/35	access	100	1G	compute-06	lot-m		
e0/36	access	100	1G	compute-08	lot-m		
e0/37	access	100	1G	compute-10	lot-m		
e0/38	access	100	1G	compute-12	lot-m		
e0/39	access	100	1G	compute-14	lom-m		
e0/40	access	100	1G	compute-16	lom-m		
e0/41	access	100	1G	compute-18	lom-m		
e0/42	access	100	1G	storage-00	lom-m		
e0/43	access	100	1G	storage-01	lom-m		
e0/44	access	100	1G	storage-02	lom-m		
e0/45	access	100	1G	storage-03	lom-m		
e0/46	access	100	1G	storage-04	lom-m		
e0/47	access	100	1G	storage-05	lom-m		
e0/48	access	100	1G	storage-06	lom-m		

Appendix B: Bills of Material

Table 7, Table 8, Table 9, and Table 10 lists the following bills of material:

- "Compute Reference Bill of Materials"
- "Storage Reference Bill of Materials"
- "Network TOR Model A (Nexus 3064) Reference Bill of Materials"
- "Network TOR Model B (Nexus 5548-UP) Reference Bill of Materials"

Table 7 Compute Reference Bill of Materials

Product	Description	Quantity
UCSC-C220-M3S	UCS C220 M3 SFF w/o CPU, mem, HDD, PCIe, PSU, w/ rail kit	1
UCS-CPU-E5-2660	2.20 GHz E5-2660/95W 8C/20MB Cache/DDR3 1600MHz	2
UCS-MR-1X162RY-A	16GB DDR3-1600-MHz RDIMM/PC3-12800/dual rank/1.35v	16
A03-D600GA2	600GB 6Gb SAS 10K RPM SFF HDD/hot plug/drive sled mounted	8
UCS-RAID-9266-NB	MegaRAID 9266-8i with no battery back up	1
R2XX-RAID10	Enable RAID 10 Setting	1
UCSC-PCIE-C10T-02	Cisco VIC 1225T Dual Port 10GBaseT CAN	1
UCSC-PSU-650W	650W power supply for C-series rack servers	2
CAB-C13-C14-2M	Power Cord Jumper, C13-C14 Connectors, 2 Meter Length	2
UCSC-DLOM-01	Dedicated LOM Mode BIOS setting for C-Series Servers	1
UCSC-HS-C220M3	Heat Sink for UCS C220 M3 Rack Server	2
UCSC-RAIL1	Rail Kit for C220, C22, C24 rack servers	1

Table 8 Storage Reference Bill of Materials

Product	Description	Quantity
UCSC-C240-M3S	UCS C240 M3 SFF w/o CPU, mem, HD, PCIe, w/ rail kit, expdr	1
UCS-CPU-E5-2609	2.4 GHz E5-2609/80W 4C/10MB Cache/DDR3 1066MHz	2
UCS-MR-1X082RY-A	8GB DDR3-1600-MHz RDIMM/PC3-12800/dual rank/1.35v	4
UCS-HDD300GI2F105	300GB 6Gb SAS 15K RPM SFF HDD/hot plug/drive sled mounted	24
UCS-RAID-9266-NB	MegaRAID 9266-8i with no battery back up	1
UCSC-PCIE-C10T-02	Cisco VIC 1225T Dual Port 10GBaseT CNA	1
UCSC-PSU-650W	650W power supply for C-series rack servers	2
CAB-C13-C14-2M	Power Cord Jumper, C13-C14 Connectors, 2 Meter Length	2
UCSC-DLOM-01	Dedicated LOM Mode BIOS setting for C-Series Servers	1
UCSC-HS-C240M3	Heat Sink for UCS C240 M3 Rack Server	2
UCSC-RAIL-2U	2U Rail Kit for UCS C-Series servers	1
UCSC-PCIF-01F	Full height PCIe filler for C-Series	3

Table 9 Network TOR Model A (Nexus 3064) Reference Bill of Materials

Product	Description	Quantity
N3K-C3064-E-FA-L3	Nexus 3064-E Std Airflow (port side exhaust) LAN Ent Lic B	1
SFP-H10GB-CU1M	10GBASE-CU SFP+ Cable 1 Meter	16
SFP-H10GB-CU3M	10GBASE-CU SFP+ Cable 3 Meter	16
N2200-PAC-400W	N2K/N3K AC Power Supply Std airflow (port side exhaust)	2
CAB-C13-C14-2M	Power Cord Jumper C13-C14 Connectors 2 Meter Length	2
N3K-BAS1K9	Nexus 3000 Base License	1
N3K-C3064-ACC-KIT	Nexus 3064PQ Accessory Kit	1
N3K-C3064-FAN	Nexus 3064 Fan Mo Front-to-Back Airflow Facing ColdAisle	1
N3K-LAN1K9	Nexus 3000 LAN Enterprise License	1
N3KUK9-602U1.1A	NX-OS Release 6.0(2)U1(1a)	1
GLC-T	1000BASE-T SFP	16

Table 10 Network TOR Model B (Nexus 5548-UP) Reference Bill of Materials

Product	Description	Quantity
N5K-C5548UP-FA	Nexus 5548 UP Chassis, 32 10GbE Ports, 2 PS, 2 Fans	1
N5548P-FAN	Nexus 5548P Fan Module	2
N55-PAC-750W	Nexus 5500 PS, 750W, Front to Back Airflow	2
CAB-C13-C14-2M	Power Cord Jumper, C13-C14 Connectors, 2 Meter Length	2
GLC-T	1000BASE-T SFP	8
SFP-H10GB-CU1M	10GBASE-CU SFP+ Cable 1 Meter	16
SFP-H10GB-CU3M	10GBASE-CU SFP+ Cable 3 Meter	8
N55-D160L3-V2	Nexus 5548 Layer 3 Daughter Card, Version 2	1
N55-M16UP	Nexus 5500 Unified Mod 16p 10GE Eth/FCoE OR 16p 8/4/2/1G FC	1
GLC-T	1000BASE-T SFP	8
SFP-H10GB-CU3M	10GBASE-CU SFP+ Cable 3 Meter	8
N5KUK9-602N1.2	Nexus 5000 Base OS Software Rel 6.0(2)N1(2)	1
N55-LAN1K9	Layer 3 License for Nexus 5500 Platform	1
N55-BAS1K9	Layer 3 Base License for Nexus 5500 Platform	1
N5548-ACC-KIT	Nexus 5548 Chassis Accessory Kit	1

Appendix C: Tested Design

The design tested is a functionally equivalent subset of the current rack level reference architecture. The test system was built out of a set of six UCS C-220-M3S and three UCS C-240-M3S servers, connected together as per Diagram 1. The 10 Gbps links provide VLAN segments for both the management plane and the primary data networks, while the 1 Gbps links provide access to the Cisco Integrated Management Controller for each node.

Summary of Test Findings

The testing described in this paper demonstrates that OpenStack is a flexible Cloud operating system that can be deployed quickly and in an automated fashion. Cisco OpenStack Installer powered by Puppet provides a flexible deployment model for OpenStack that allows cloud administrators to deploy OpenStack networking, compute, storage, and identity services on multiple compute nodes with minimal effort. Test cases described in Appendix C: Tested Design include:

1. OpenStack Installation on Multiple Nodes, page -19
2. OpenStack Installation with GRE Tunnels Used for Isolation, page -23
3. OpenStack Installation Automated with Minimal Input, page -24
4. Monitoring Functional State with An Administrative Dashboard, page -25
5. OpenStack Manual Installation Minimum Deployment Time, page -27
6. OpenStack Application Infrastructure Deployment Minimum Deployment Time, page -28

OpenStack Test Cases

To highlight functional use cases, the following tests were performed:

OpenStack Installation on Multiple Nodes

- **Nr**: 001
- **Name**: Deployment
- **User**: System Administrator
- **Case**: Install

It should be possible to deploy OpenStack in a multi-node fashion on physical compute, network, and storage elements

Test Setup

The topology for this test includes six UCS C-220-M3S and three UCS C-240-M3S servers connected to a Nexus 3064 top-of-rack switch pair. The first C220 will serve as a “build” node to serve DHCP/PXE and Puppet for installing the other servers. The second C220 will serve as the control node on which coordinating services for the cloud will be run (including Horizon, the OpenStack GUI). The next three C220s serve as compute nodes on which instances may be launched. The last C220 serves as the swift proxy node. All three C240s serve as swift storage nodes.

Each server is equipped with two 10-gigabit Ethernet interfaces. Each of the two 10-gigabit Ethernet interfaces is cabled to a separate Nexus 3064 top-of-rack (ToR) switch. All server ports on the ToR switches should be configured as trunk ports, with management network being the native VLAN on the trunk:

```
interface Ethernet1/17
  switchport mode trunk
  switchport trunk native vlan 844
  switchport trunk allowed vlan 842,844,846
  spanning-tree port type edge trunk
```

Procedure

Step 1 Install Ubuntu 12.04 Server on the first node in the cluster (e.g., the "build node").

Step 2 As root, run the following commands to apply updates and install a few necessary software packages:

- **a.** apt-get update
- **b.** apt-get dist-upgrade –y
- **c.** apt-get install git ipmitool debmirror

Step 3 Fetch the Cisco OpenStack Installer – Grizzly Puppet modules by running the following command as root:

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

Step 4 Copy the site.pp.example file to site.pp and edit it to customize it for your environment. You will supply information about the remaining nodes in your cloud, networking information, and other configuration information.

- **a.** cp site.pp.example site.pp
- **b.** vi site.pp

Step 5 If necessary, edit cobbler-node.pp to add any late commands you would like to use to customize your bare-metal operating system installs.

Step 6 Apply the edited site.pp file. Once the site.pp file has been applied, make puppet plugins available for nodes to download:

- **a.** puppet apply –v site.pp
- **b.** puppet plugin download

Step 7 Install the control node by issuing the following commands to power cycle it. When the machine reboots, it should begin a PXE installation of Ubuntu 12.04. You may wish to launch the KVM console from the CIMC before issuing these commands so you can observe the process as it takes place.

- **a.** cobbler system poweroff controller
- **b.** cobbler system poweron controller

Step 8 When the control node has completed the operating system install, log in to it as localadmin with password "ubuntu."

Step 9 If you disabled puppet from running at startup via your site.pp file to make any final changes before running the installation process:

- **a.** Make your final changes.
- **b.** Edit /etc/default/puppet and change "no" to "yes,"

c. Then run the following command as root:

```
service puppet start
```

Step 10 Observe the puppet agent logging messages in /var/log/syslog as it installs and configures the OpenStack control node. You should not see any error messages.

Step 11 Once the control node has completed its installation process, boot the compute nodes in the cluster. As they boot, they will begin PXE installation of Ubuntu 12.04. You may wish to access the KVM consoles of each system prior to issuing these commands in order to observer the install process.

a. cobbler system poweroff compute-01; cobbler system poweron compute-01

b. cobbler system poweroff compute-02; cobbler system poweron compute-02

c. cobbler system poweroff compute-03; cobbler system poweron compute-03

Step 12 Once the operating system install process completes, log in to each node as localadmin with password "ubuntu."

Step 13 If you chose to disable puppet from running at startup via your site.pp file to make any final changes to each node before starting the OpenStack installation process:

a. Make your final changes

b. Start the puppet agent by running the following command as root on each node:

```
service puppet start
```

Step 14 Observe the puppet log messages in /var/log/syslog on each system. You should not see any errors.

Step 15 Once the compute nodes have finished their automated setup process, boot the swift storage nodes in the cluster. As they boot, they will begin PXE installation of Ubuntu 12.04. You may wish to access the KVM consoles of each system prior to issuing these commands in order to observer the install process.

a. cobbler system poweroff swift-storage01; cobbler system poweron swift-storage01

b. cobbler system poweroff swift-storage02; cobbler system poweron swift-storage02

c. cobbler system poweroff swift-storage03; cobbler system poweron swift-storage03

Step 16 Once the operating system install process completes, log in to each node as localadmin with password "ubuntu."

Step 17 If you chose to disable puppet from running at startup via your site.pp file to make any final changes to each node before starting the OpenStack installation process:

a. Make your final changes

b. Start the puppet agent by running the following command as root on each node:

```
service puppet start
```

Step 18 Observe the puppet log messages in /var/log/syslog on each system. You should not see any errors.

Step 19 Once the swift storage nodes have finished their automated setup process, boot the swift proxy node in the cluster. As it boots, it will begin PXE installation of Ubuntu 12.04. You may wish to access the KVM console of the system prior to issuing these commands in order to observer the install process.

a. cobbler system poweroff swift-proxy01

b. cobbler system poweron swift-proxy01

Step 20 Once the operating system install process completes, log in to it as localadmin with password "ubuntu."

Step 21 If you chose to disable puppet from running at startup via your site.pp file to make any final changes to each node before starting the OpenStack installation process:

a. Make your final changes

b. Start the puppet agent by running the following command as root on each node:

```
service puppet start
```

Step 22 Observe the puppet log messages in /var/log/syslog on each system. You should not see any errors.

Step 23 Once the swift proxy node has finished its automated setup process, run the following command on the control node and verify that each compute node is listed in the output:

a. nova-manage host list

Step 24 Access Horizon by opening a web browser and typing in the IP address or hostname of your control node. Verify that you can log in as "admin" with password "Cisco123."

Step 25 Download VM images to the control node and import into Glance with the following commands:

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

b. glance add name="precise-x86_64" is_public=true container_format=ovf disk_format=qcow2 < precise-server-cloudimg-amd64-disk1.img

c. wget http://mattdm.fedorapeople.org/cloud-images/Fedora18-Cloud-x86_64-latest.qcow2

d. glance add name="Fedeora18-x86_64" is_public=true container_format=ovf disk_format=qcow2 < Fedora18-Cloud-x86_64-latest.qcow2

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

f. glance add name="cirros-x86_64" is_public=true disk_format=qcow2 container_format=ovf < cirros-0.3.1-x86_64-disk.img

Step 26 Setup quantum networking including:

a. Public network with a pool for floating IPs

b. Private network with a sunet for VM hosts

c. Quantum router connected to both networks

Step 27 Using CLI or UI, setup two new tenants with a non-admin user in each tenant.

Step 28 Install tempest on control node and customize the temptest.conf file.

a. git clone https://github.com/CiscoSystems/tempest.git

b. cd tempest

c. git checkout -b stable/grizzly origin/stable/grizzly

d. pip install testtools

e. pip install testresources

f. vi etc/tempest.conf

Step 29 Run tempest using nose test:

```
nosetests -v tempest
```

Pass/Fail Criteria

1. Puppet should run without errors that prevent OpenStack from functioning.
2. User should be able to log in to Horizon.
3. User should be able to successfully create Quantum routers, networks, and subnets.
4. Users should be able to import VM images into glance

5. Users should be able to create new tenants and users in each tenant.
6. Tempest should be able to run and tests pass.

Bugs Encountered

- https://bugs.launchpad.net/tempest/+bug/1202991
- https://bugs.launchpad.net/neutron/+bug/1190242
- https://bugs.launchpad.net/tempest/+bug/1160309
- https://bugs.launchpad.net/nova/+bug/1197573
- https://bugs.launchpad.net/tempest/+bug/1182384
- https://bugs.launchpad.net/tempest/+bug/1198164
- https://bugs.launchpad.net/nova/+bug/1189059

OpenStack Installation with GRE Tunnels Used for Isolation

- **Nr**: 002
- **Name**: Network Deployment
- **User**: System Administrator
- **Case**: Install

It should be possible to deploy OpenStack using the following network model:

Multiple physical network interfaces for management and tenant or other networks with GRE tunnels used for isolation on one or more virtual or physical OpenStack elements.

Test Setup

Topology deployment with GRE Isolation is documented at:

http://docwiki.cisco.com/wiki/OpenStack:Grizzly-Multinode

We use a 9-node topology as described in OpenStack Installation on Multiple Nodes, page -19.

Procedure

Testing requires at least 3 servers connected by a ToR switch.

Step 1 Install a minimal install of Ubuntu 12.04 LTS on the first server (the “build” server)

Step 2 Add the git and puppet and ipmitool packages to the build server

Step 3 curl -s -k -B https://raw.github.com/CiscoSystems/grizzly-manifests/multi-node/install_os_puppet | /bin/bash

Step 4 cd /etc/puppet/manifests

Step 5 edit site.pp as appropriate for your install

Step 6 sh puppet-modules.sh

Step 7 run puppet to configure your build node: puppet apply –v –d /etc/puppet/manifests/site.pp

Step 8 puppet plugin download

Step 9 cobbler system list to verify

Step 10 ./clean_node.sh controller to install the controller

Step 11 After the controller installs and reboots, log into it and monitor the automatic puppet run that sets it up

Step 12 Once it finishes, repeat steps 10-11 for the compute nodes

Step 13 Once they finish, repeat steps 10-11 for the swift storage nodes

Step 14 Once they finish, repeat steps 10-11 for the swift proxy nodes

Step 15 Run puppet agent –t –d on the build node to finish configuring Nagios

Pass/Fail Criteria

Tempest was used to validate the OpenStack network.

See results from OpenStack Installation on Multiple Nodes, page -19.

OpenStack Installation Automated with Minimal Input

- **Nr**: 003
- **Name**: Deployment Automation
- **User**: System Administrator
- **Case**: Install

The installation should run automatically once basic information is provided to the installer. This information should not need to exceed the following:

1. Network address information
2. Physical device address and access information (e.g., power management, MAC address)
3. User names and passwords if necessary for external or API access
4. Deployment model information (network interface bonding, disk use models, etc.)
5. Network and storage control interfaces for external devices
6. Default management information for automatically deplorable network and storage elements

Test Setup

See OpenStack Installation on Multiple Nodes, page -19. This test case requires only the prerequisites for whichever OpenStack installation method you choose.

Procedure

Follow the instructions in OpenStack Installation on Multiple Nodes, page -19 to obtain a copy of site.pp.example.

All user-configurable install instructions reside in site.pp.example, which after editing, must then be copied to site.pp for execution.

site.pp.example is well-documented internally. The general information that needs to be known to correctly modify the file is as follows:

- Whether or not you need to use a proxy to reach the Internet. For convenience, Cisco offers the options of both HTTP and FTP for package retrieval.
- An accessible NTP server.

- The general network configuration you wish to use:
- The IP addresses, hostnames, and domain names you wish to use for your build, control, and compute nodes.
- The physical hardware in each node that will connect to the respective physical/logical network (e.g., network interfaces).
- The MAC addresses for each node.
- The netmask and default gateway used for the private network.
- IP and login information for power control of the physical nodes.

By default, Cisco pre-populates most of this information. Cisco also defines one of each node: build, controller, and compute. To add additional nodes, you must add additional node information as described in site.pp.example.

Complete the installation process described in OpenStack Installation on Multiple Nodes, page -19.

Pass/Fail Criteria

The test case passes if the installation procedure in OpenStack Installation on Multiple Nodes, page -19 section 1.1 can be completed without adding additional information to site.pp beyond the node information described in the procedure.

Monitoring Functional State with An Administrative Dashboard

- **Nr**: 004
- **Name**: Management and Monitoring
- **User**: System Administrator
- **Case**: System Functional State

The functional state of the system should be available to a systems state dashboard of some nature allowing for a view of areas of function or areas where a system maintainable or repair task is required. Failures should be reported in a time commensurate with their severity. Capacity limitations and system bottlenecks should also be discoverable via these mechanisms.

It should be possible to see:

- Control
  - Total server down alerts
- Compute
  - Total server down alerts
- Network
  - Total server down alerts
  - Rabbit messaging alerts are working
  - Rabbit API alerts are working
- System
  - Total server down alerts
  - Keystone service not running alerts
  - Keystone authentication not working alerts

- Keystone endpoints not defined alerts
- NTP times not sufficiently synchronized alerts
- nova-api service not running alerts
- nova-api service not answering queries alerts
- glance-api not running alerts
- glance-api misconfigured alerts
- glance alerts if no images found
- apache for Horizon not running triggers alerts
- VM status logs cpu / memory consumption are gathered

Test Setup

Testing should be done with a standard multi-node test pod. The pod will be installed using puppet in the standard Cisco OpenStack Installer multinode deployment as documented at:

http://docwiki.cisco.com/wiki/OpenStack:Grizzly-Multinode

To prepare for testing, configure puppet with information about the build node, the control node, and two compute nodes. Then, deploy only the build node.

Procedure

Step 1 Deploy the control node. After the puppet run finishes, power off the node and verify Nagios alerting about control node.

Step 2 Deploy the compute node. After the puppet run finishes, power off the node and verify Nagios alerting about compute node.

Step 3 When executing Step 1, verify that network connectivity Nagios alerts are triggered.

Step 4 Stop the rabbitmq-server service on the controller. Verify “RabbitMQ Alive” Nagios alerts triggered.

Step 5 When executing Step 1, verify that system tests (load, users, disk space) generate Nagios alerts.

Step 6 Stop the keystone service on the control node and verify the Keystone alert is triggered.

Step 7 Configure an incorrect password in /etc/nrpe.d/check_keystone.cfg on the control node and verify the Keystone alert is triggered.

Step 8 Delete endpoints for services from Keystone and verify the Keystone alert is triggered.

Step 9 Stop NTP services throughout the cluster and change a system’s clock by 10 minutes. Confirm that Nagios NTP alerts are triggered by the clock drift.

Step 10 Stop the nova-api service on the controller and verify that the Nova-Api alert is triggered.

Step 11 Configure an incorrect password in /etc/nrpe.d/check_novaapi.cfg on the control node and verify the Nova-Api alert is triggered.

Step 12 Stop the glance-api service on the controller and verify that the Glance alert is triggered.

Step 13 Configure an incorrect password in /etc/nrpe.d/check_glance.cfg on the control node and verify the Glance alert is triggered.

Step 14 Confirm that the Glance alert logs errors until an image called “precise-x86_64” has been uploaded.

Step 15 Stop the apache2 service on the control node and verify the HTTP alert is triggered.

Step 16 Launch a couple of VMs and verify that CPU / memory statistics are logged under VM Stats alerts.

Pass/Fail Criteria

The test case passes if all steps above succeed.

Bugs Encountered

https://bugs.launchpad.net/openstack-cisco/+bug/1200316

https://bugs.launchpad.net/openstack-cisco/+bug/1200454

OpenStack Manual Installation Minimum Deployment Time

- **Nr**: 005
- **Name**: System Deployment Times
- **User**: System Administrator
- **Case**: Deployment Times

The system should deploy rapidly. The following are target times that should be possible to meet:

- Initial setup of a node from scratch, assuming all addressing and other manual entry information exists: 120 minutes
- Additional node setup: 60 minutes
- Network setup: 60 minutes
- Additional network setup: 60 minutes
- Storage environment setup: 120 minutes
- Additional storage setup: 120 minutes

Test Setup

Refer to setup in OpenStack Installation on Multiple Nodes, page -19. A multimode deployment model will be used in this test.

Procedure

Follow the manual install instructions for Cisco OpenStack Installer located here:

http://docwiki.cisco.com/wiki/OpenStack:Grizzly-Multinode

- Results in installation times of the following (approximately):
- Controller Node: 25 minutes
- Network Node: 20 minutes
- Compute Node: 20 minutes
- Storage Nodes: 46 minutes
- Openstack testing, including creation of virtual networks, instances, and instances: < 1 minute
- Total time: 111 minutes

Pass/Fail Criteria

The test fails if the times specified in the test description are not achievable.

OpenStack Application Infrastructure Deployment Minimum Deployment Time

- **Nr**: 006
- **Name**: Application Deployment
- **User**: Systems User
- **Case**: Application Deployment Times

An application infrastructure should deploy rapidly. This case is not intended to include application code deployment as part of the definition.

- Deployment of a VM instance from cli or UI: 10 minutes
- Deployment of network from cli or UI: 10 minutes
- Deployment of additional volume from cli or UI: 10 minutes
- Deployment of object storage container from cli or UI: 10minutes
- Notification of completion of an operation: 10 minutes

Test Setup

A standard multi-node installation of Cisco OpenStack Installer as described in OpenStack Installation on Multiple Nodes, page -19 is used for this test.

Requirements:

- 1 COSI Build Node Instance. Dedicated hardware.
- 1 COSI Control Node Instance. Dedicated Hardware.
- (at least) 1 COSI Compute Node Instance. Dedicated Hardware.
- 1 COSI Swift Proxy Node Instance. Dedicated Hardware
- (at least) 3 COSI Swift Storage Node Instances. Dedicated Hardware
- 1 Cirros testing machine image shall be loaded and available via either installation through Glance and or the horizon dashboard.
- Environment should be pre-qualified for proper operation of python command line client tools and Horizon dashboard.
- No instances shall be running at the start of the test.

Procedure

Deployment from CLI in less than 10 minutes.

CLI Deployment scripts are available with elapsed timer for job submission. This alone, however, will not show accurate elapsed time as OpenStack is an asynchronous system. You must watch for proper transition of the instance availability via the nova list command.

Step 1 Create instance using the nova boot command and watch status using the nova list command

Follow this simple example for specification of the test procedure:

Step 2 nova boot –image <image name> -- flavor<flavor name> <instance name>.

Step 3 Check command submission elapsed time and watch for instance to move to an active state by running the nova list.

Step 4 After a period of 10 minutes, the instance should be in active state.

If an error is encountered or the instance never reaches state 'ACTIVE' then declare test failed.

Deployment of network in less than 10 minutes.

CLI network deployment scripts are available with elapsed timer for job submission. This alone, however, will not show accurate elapsed time as OpenStack is an asynchronous system. You must watch for proper network availability via the quantum net-show command.

Step 1 Create network using the quantum net-create command and watch status using the quantum net-show command

Follow this simple example for specification of the test procedure:

Step 2 quantum net-create private

Step 3 Command should complete within 10 minutes

Step 4 quantum net-show should list network private

If an error is encountered running the quantum net-create command, the test result is a failure

Deployment of volume less than 10 minutes.

CLI volume scripts are available with elapsed timer for job submission. This alone, however, will not show accurate elapsed time as OpenStack is an asynchronous system. You must watch for proper volume availability via the cinder list command.

Step 1 Create 1GB cinder volume and watch status using the cinder list command

Follow this simple example for specification of the test procedure:

Step 2 cinder create –display_name test 1

Step 3 Command should complete within 10 minutes

Step 4 cinder list

If an error is vencountered running the cinder commands, the test result is a failure.

Deployment of object into container less than 10 minutes.

For this test we will up upload a 10Mb file into a new swift container and verify it completes within the time limits allowed.

Follow this simple example for specification of the test procedure:

Step 1 dd if=/dev/zero of=random.10Mb bs=10485760 count=1; du -sm random.10Mb

Step 2 The following command should complete within 10 minutes

```
date;swift upload test_container random.10Mb;date
```

If an error is encountered running the swift commands, the test result is a failure.

Pass/Fail Criteria

Instance Create

Pass: Instance shows ACTIVE via the Nova List Command

Fail: Instance shows status other than ACTIVE, command error or exceeds 10 minutes.

Network Create

Pass: quantum net-show command shows the created network

Fail: quantum net-show lists nothing, command error or exceeds 10 minutes.

Volume Create

Pass: cinder list command shows the created 1 GB volume

Fail: cinder list command lists nothing, command error or exceeds 10 minutes.

Object Container Create

Pass: swift upload command completes without error in less than 10min.

Fail: swift upload command returns an error or runtime exceeds 10 minutes.