Reference Architecture for OpenStack Grizzly with Red Hat RDO

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 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. On top of this Reference Architecture, OpenStack has been deployed using Red Hat RDO. RDO uses a community support model similar to Fedora and is not supported by Red Hat directly. RDO utilizes packstack, a puppet based system configuration tool, to deploy OpenStack and its related tools across the servers making up the reference architecture.

Reference Architecture

The reference architecture is intended to support a common model for OpenStack deployments, meeting an average 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.

While the reference architecture is intended to be able to support all core services in the current OpenStack project, minor gaps exist in support between the core project and the current tested reference. Specifically, due to Cisco's commitment to efficiently delivering value to its customers, no fine-tuning of component setup outside of the default packstack installation has been performed.

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:

- Ability to support highly available access to OpenStack services (Nova, Glance, Keystone, Quantum, Swift, and Cinder).
- 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 for bonded physical network interface resilience.
- Support OpenStack Grizzly releases against Red Hat 6.4 and its derivatives using RDO installed via packstack.

As noted above, the model and our validation diverge in areas where we do not yet have support from the Cisco Reference Architecture testing and validation efforts.

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.

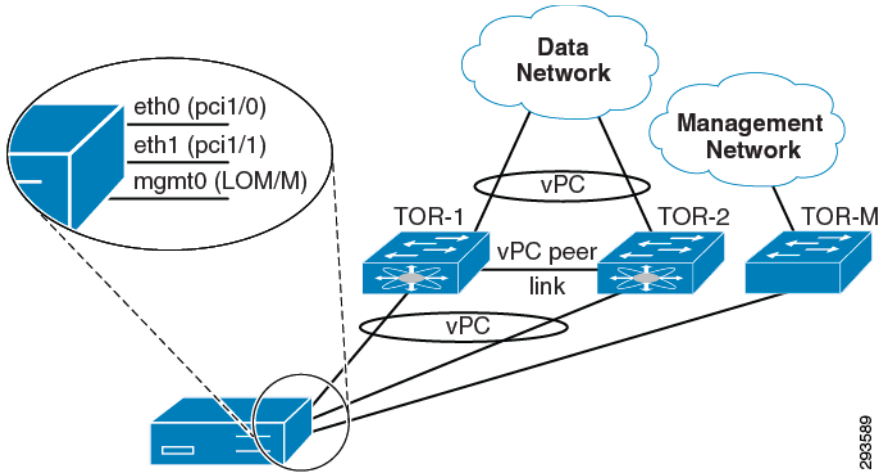
A set of utility scripts are available to facilitate BIOS updates and storage configurations, along with collecting data needed for the reference deployment model. These scripts are available from the Cisco Systems GitHub repository:

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:

- VLAN 100 is 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 ports by the Cisco Quantum plugin.
- vPC is used to connect to an upstream router (or routers) who support LACP port channels at a minimum.

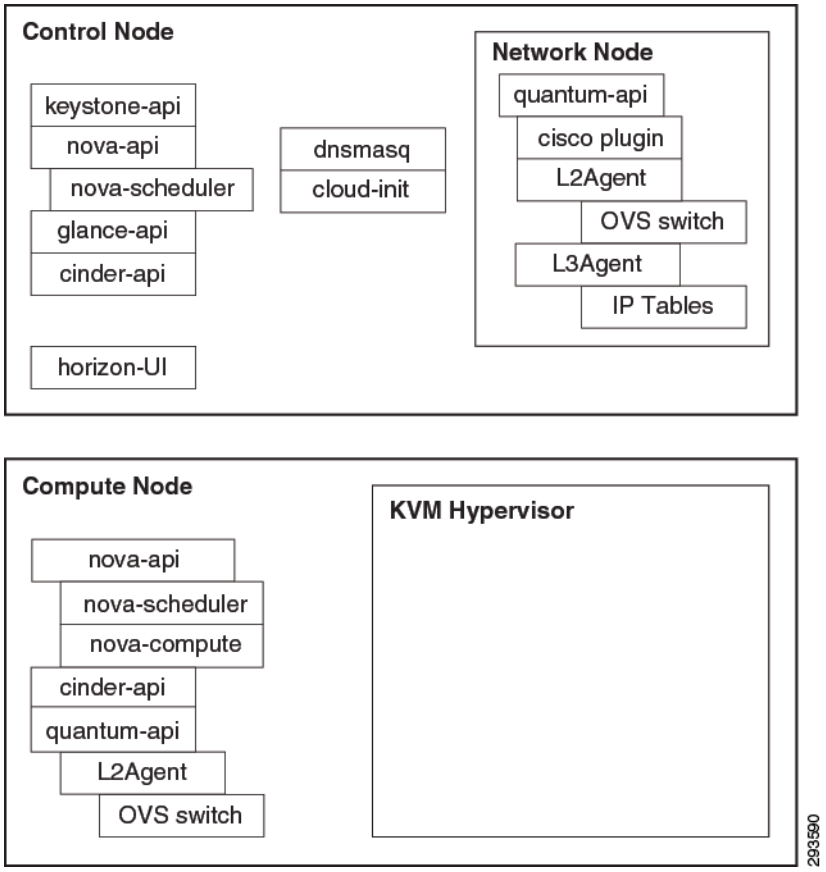
Logically, the network is segregated either via VLANs (as proposed in this reference) or via an overlay L2 technology like VXLAN. The latter is expected to become the standard mechanism for providing much greater tenancy or project-level L2 segregation scale by the Fall 2013 OpenStack release time frame (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 -19.

Software Architecture

The system software architecture for the Grizzly release of RDO on CentOS 6.4 is straightforward (Figure 2). 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.

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 in order 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	Blank
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 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. It should be able to provide on the order of ~480 virtual machines and has both scalable block and object storage architectures.

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
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

Table 5 Rack Scale Example System (continued)

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.

The version of Open vSwitch (OVS) provided with RDO does not support GRE-tunneling, so tenant segmentation must be performed using provider VLANs. The controller and every compute node must have an interface that is connected to a shared L2-network and is able to trunk the set of VLANs assigned to the cluster. The control node additionally needs connectivity to the public network from which floating IPs will be assigned.

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 CentOS 6.4 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 address.

Also, when partitioning the storage, choose a partitioning scheme that provides at least 15 GB free space under /var, as installation packages and ISO images used to deploy OpenStack will eventually be cached there.

Note If you have proxies, or your control and compute nodes do not have Internet access, 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.
- If you do have a proxy, you will want not only to export the two types of proxies needed in your root shell when running fetch commands but also update the yum.conf file to include the proxy configuration.

Cobbler Installation

Before beginning cobbler installation, we want to update all the packages in the system to make sure they are current, and the EPEL repository, and then install cobbler and other necessary packages. Open a root shell on the build server and issue the following commands:

```
# yum -q -y update
```

```
# yum -q -y install
http://dl.fedoraproject.org/pub/epel/6/x86_64/epel-release-6-8.noarch.rpm
# yum -y -q install cobbler dnsmasq pykickstart fence-agents
```

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

Cobbler and SELinux don't play well together. The workaround we utilized is to turn SELinux from the default enforcement mode to permissive mode. Edit the /etc/selinux/config file and change the SELINUX= directive from enforcement to permissive. Then reboot.If your environment requires SELinux to be in enforcement mode, there are some hints to making it work at this URL:

https://github.com/cobbler/cobbler/wiki/Selinux

Customizing the Build Server

Once the packages are installed, we'll need to customize the configuration for our environment. In our network, there isn't a DHCP server already setup for our VLANs. If your setup already has a DHCP server, some of the configuration shown will need to be performed on your already existing DHCP server instead of the build server.

Step 1 First generate an encrypted password to use for the root use on all the servers in the cluster.

```
# openssl passwd -1
Password:
Verifying - Password:
$1$4Wzk2nuH$KruDeqkr9E7F7gBanYFYc
```

Step 2 Next edit the /etc/cobbler/settings file to include the generated password and the necessary DNS, DHCP, and PXE settings. In this example, 172.29.84.70 is the IP address of the build server we are configuring.

```
# vi /etc/cobbler/settings
  default_password_crypted: "$1$4Wzk2nuH$KruDeqkr9E7F7gBanYFYc."
  Manage_dhcp: 1
  manage_dns: 1
  next_server: 172.29.84.70
  pxe_just_once: 1
  server: 172.29.84.70
```

Step 3 Since we'll be using some bleeding edge packages after the install, we don't want the local cobbler server as yum repo.

```
# vi /etc/cobbler/settings
  yum_post_install_mirror: 0
```

Step 4 Dnsmasq is a combined DNS and DHCP server popular in the OpenStack community. Configure cobbler to utilize dnsmasq, and then customize dnsmasq for the topology. In this example, 8.8.8.4 is the upstream DNS server, and 172.29.84.65 is the default router for the VLAN.

```
# vi /etc/cobbler/modules.conf
  [dns]
  module = manage_dnsmasq
  [dhcp]
  module = manage_dnsmasq

# vi /etc/cobbler/dnsmasq.template
  server=8.8.8.4
```

```
  dhcp-range=172.29.84.71,172.29.84.99
  dhcp-option=3,172.29.84.65

Enable and then start the required services on the build server.

# chkconfig rsync on
# chkconfig tftp on
# chkconfig dnsmasq on
# chkconfig httpd on
# chkconfig xinetd on
# chkconfig cobblerd on
# service dnsmasq start
# service httpd start
# service xinetd start
# service cobblerd start
```

Step 5 Enable and then start the required services on the build server.

```
# chkconfig rsync on
# chkconfig tftp on
# chkconfig dnsmasq on
# chkconfig httpd on
# chkconfig xinetd on
# chkconfig cobblerd on
# service dnsmasq start
# service httpd start
# service xinetd start
# service cobblerd start
```

Step 6 Open up firewall ports on the build server to allow the services to be reachable.

- DNS

```
# iptables -I INPUT 5 -m state --state NEW -m tcp -p tcp --dport 53 -j ACCEPT
# iptables -I INPUT 5 -m state --state NEW -m udp -p udp --dport 53 -j ACCEPT
```

- DHCP & TFTP

```
# iptables -I INPUT 5 -m state --state NEW -m udp -p udp --dport 68:69 -j ACCEPT
```

- HTTP/S

```
# iptables -I INPUT 5 -m state --state NEW -m tcp -p tcp --dport 80 -j ACCEPT
# iptables -I INPUT 5 -m state --state NEW -m tcp -p tcp --dport 443 -j ACCEPT
```

- Cobbler

```
# iptables -I INPUT 5 -m state --state NEW -m tcp -p tcp --dport 25150:25151 -j ACCEPT
```

Step 7 And then save the current iptables so they will be reloaded correctly on reboot.

```
# service iptables save
```

Step 8 Cobbler provides a utility to check for any missing pieces or config. Run the check and fix anything that looks relevant to your setup. Then restart the cobbler service and run 'cobbler sync' to push the configuration out to the other services (dnsmasq, etc).

```
# cobbler check
# service cobblerd restart
# cobbler sync
```

Customizing the Installer

To customize the installer, perform the following:

Step 1 Import CentOS 6.4 distribution for cobbler to install onto the other servers in the cluster. Since Cisco UCS uses 64-bit processors we need import x86-64 architecture. Pick a CentOS mirror close to your server from the official mirror list, or you can import from DVD.

```
# cobbler import --name=CentOS_6.4
--path=rsync://mirror.stanford.edu/mirrors/centos/6.4/os/x86_64 --arch=x86_6
```

Step 2 The next piece is to tie the distribution together with a kickstart file in what cobbler calls a profile. When you import a distribution, a default profile is created for that distribution using the sample.ks kickstart file. If you want more control of the install process, you can customize the kickstart file and create a new profile. For a simple RDO installation the sample.ks file is sufficient.

Note Servers installed using the sample.ks files will have selinux disabled. If you use a different install method, you'll want to make sure selinux is either disabled or in permissive mode.

Step 3 The last piece is to tie the profile together with the information specific to each server in our cluster into what cobbler calls systems. For each system, specify the hostname and the profile you want to use to kickstart it. Optionally specifying the IP of CIMC as well as username and password will allow cobbler to reboot it.

```
# cobbler system  add --name=pod3-server11 --hostname pod3-server11
--profile=CentOS_6.4-x86_64 --netboot-enabled=true --power-address=172.29.84.11
--power-user admin --power-pass password --power-type=ipmilan
```

Step 4 Next tell cobbler how the network for this server is to be configured. In our topology, eth2 gets and IP address for kickstart and management, and eth3 needs to be up but doesn't require an IP address since it will be the interface used for tenant VLANs.

```
# cobbler system edit --name=pod3-server11 --interface=eth2 --mac=44:03:a7:c5:96:81
--ip-address=172.29.84.71 --subnet=255.255.255.192 --gateway=172.29.84.65 --static=1
# cobbler system edit --name=pod3-server11 --interface=eth3 --mac=44:03:a7:c5:96:82
--static=1
```

Step 5 Add a system for every device in your cluster. Do another 'cobbler sync' and then have cobbler reboot each server install CentOS 6.4 for you!

```
# cobbler sync
# for SERVER in `cobbler system list`; do cobbler system edit --netboot-enabled=true
--name=$SERVER; cobbler system reboot --name=$SERVER; done
```

Packstack Installation of RDO

Now that all servers in the cluster have CentOS 6.4 installed, we can begin to deploy RedHat RDO on top.

Step 1 Open a root shell on the control node and install packstack.

```
# yum install -q -y
http://rdo.fedorapeople.org/openstack/openstack-grizzly/rdo-release-grizzly-3.noarch.r
pm
```

```
# yum install -q -y openstack-packstack
```

Step 2 Now that packstack is installed on the control node, we'll tell it to deploy RDO on the entire topology using the –install-hosts= directive. The first host in the list will become your control node, and the rest will become compute nodes. The same server on which we installed packstack we'll specify to be our control node. The rest of the servers will be compute nodes. We'll also specify to use VLAN tenant segregation, to use VLANS 4048-4093 on the private network, and the bridge to use to connect to the private network will be called br-eth3.

Note You'll have to type the root password for each host to allow packstack to start the SSH connection.

```
# packstack
--install-hosts=172.29.84.71,172.29.84.72,172.29.84.73,172.29.84.74,172.29.84.75,1
72.29.84.76,172.29.84.77,172.29.84.78,172.29.84.79,172.29.84.80,172.29.84.81,172.2
9.84.82,172.29.84.83,172.29.84.84,172.29.84.85,172.29.84.86,172.29.84.87,172.29.84
.88,172.29.84.89 --quantum-ovs-tenant-network-type=vlan
--quantum-ovs-vlan-ranges=physnet2:4048:4093
--quantum-ovs-bridge-mappings=physnet2:br-eth3
```

If the packstack installation does not complete successfully, check the log or error messages and fix any problems found. If you hit the IPtables error shown below, it is due to a race condition when doing multi-node install. Its documented in RedHat Bugzilla 969869.

https://bugzilla.redhat.com/show_bug.cgi?id=969869

```
ERROR : Error during puppet run : err: /Firewall[001 nova compute
incoming]/ensure: change from absent to present failed: Execution of
'/sbin/iptables -I INPUT 1 -t filter -p tcp -m multiport --dports 5900:5999 -m
comment --comment 001 nova compute incoming -j ACCEPT' returned 4: iptables:
Resource temporarily unavailable.
Please check log file
/var/tmp/packstack/20130602-214504-2DIMwC/openstack-setup.log for more information

Additional information:
 * A new answerfile was created in: /root/packstack-answers-20130602-214505.txt
 * Time synchronization installation was skipped. Please note that unsynchronized
time on server instances might be problem for some OpenStack components.
 * To use the command line tools you need to source the file
/root/keystonerc_admin created on 172.29.84.71
 * To use the console, browse to http://172.29.84.71/dashboard
```

Once you fix any problems, you can re-run the installation over again using the answer (aka config) file produced by packstack during the install.

```
# packstack --answer-file=/root/packstack-answers-20130602-214505.txt
```

Step 3 Once packstack install completes successfully, your OpenStack cloud is up and running. You can login to Horizon using the credentials located in the /root/keystonerc_admin file, or source the file from a root shell and configure from the command line.

Post Install Configuration

Packstack will install a new kernel onto each node in the cluster. Before we reboot, however, we need to do a little bit of manual configuration.

Step 1 First we need to configure the quantum agents to use virtual Ethernet.

```
# sed -i "s/^[# ]*ovs_use_veth.*$/ovs_use_veth = True/g" /etc/quantum/l3_agent.ini
# sed -i "s/^[# ]*ovs_use_veth.*$/ovs_use_veth = True/g" /etc/quantum/lbaas_agent.ini
# sed -i "s/^[# ]*ovs_use_veth.*$/ovs_use_veth = True/g" /etc/quantum/dhcp_agent.ini
```

Step 2 Next we'll need to add public interface information to the OVS plugin configuration on the control node. Edit the /etc/quantum/plugins/openvswitch/ovs_quantum_plugin.ini file to add configuration for physnet1:

```
network_vlan_ranges=physnet1,physnet2:4048:4093
bridge_mappings=physnet1:br-ex,physnet2:br-eth3
```

Step 3 When the control server was kickstarted, it used IP connectivity on the physical interface connected to the public network to download and install packages. We'll need to move this IP connectivity to OVS bridge and make the physical interface a port on that bridge. First, copy the current physical interface network config script to the new bridge interface config script.

```
# cp /etc/sysconfig/network-scripts/ifcfg-eth2
/etc/sysconfig/network-scripts/ifcfg-br-eth2
```

Step 4 Then edit the bridge interface config script to remove physical attributes (HWADDRESS), add OVS attributes (DEVICE, TYPE, & DEVICETYPE), and retain the IP address information.

```
DEVICE=br-eth2
TYPE=OVSBridge
DEVICETYPE=ovs
ONBOOT=yes
BOOTPROTO=static
IPADDR=172.29.84.71
NETMASK=255.255.255.192
```

Step 5 Finally, edit the physical interface config script to remove IP address information (IPADDR & NETMASK) and add OVS attributes (DEVICETYPE, TYPE, OVS_BRIDGE).

```
DEVICE=eth2
ONBOOT=yes
HWADDR=44:03:A7:C5:96:81
TYPE=Ethernet
DEVICETYPE=ovs
BOOTPROTO=none
TYPE=OVSPort
OVS_BRIDGE=br-eth2
```

Step 6 Now reboot all the nodes in the cluster to the kernel and network configuration. Once all the nodes in the cluster are back online, we'll need to add the physical interfaces to the appropriate OVS bridges. In our setup all the nodes use Eth3 to access VLANs 4048-4093 which implement the private networks. When we ran packstack, we specified br-eth3 to be the ovs bridge, so on each node we need to add eth3 to br-eth3.

Note Perform this command on every node in the cluster.

```
# ovs-vsctl add-port br-eth3 eth3
```

Step 7 In our setup, the control node users Eth2 to access the public network on VLAN844. Patch the br-eth2 OVS bridge we setup just prior to booting the new kernel into the br-ex bridge that quantum uses for public network access. Use the tag=844 parameter when creating the patch port on br-eth2.

Note Perform these commands on only the control node in the cluster.

```
# ovs-vsctl add-port br-eth2 phy-br-eth2-844 tag=844
# ovs-vsctl set interface phy-br-eth2-844 type=patch
# ovs-vsctl set interface phy-br-eth2-844 options:peer=ex-br-eth2
```

Step 8 Next create the corresponding patch port on br-ex.

```
# ovs-vsctl add-port br-ex ex-br-eth2
# ovs-vsctl set interface ex-br-eth2 type=patch
# ovs-vsctl set interface ex-br-eth2 options:peer=phy-br-eth2-844
```

Step 9 Finally, if you are using Cisco Virtual Interface Card (VIC), you also need set OVS to treat the physical interfaces as trunks and turn on VLAN splinters. This is to work around a bug in RHEL 6.4 and derivatives which causes the VLAN tags get dropped when received by other nodes sitting on the network if VLANs are being used to isolate tenant networks.

```
# ovs-vsctl set port eth2 trunks=0,844
# ovs-vsctl set interface eth2 other-config:enable-vlan-splinters=true
```

Note Perform this command on every node in the cluster.

```
# ovs-vsctl set port eth3 trunks=`seq -s , 4048 4093`
# ovs-vsctl set interface eth3 other-config:enable-vlan-splinters=true
```

Validation: Deploy 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 --tenant-id services
--provider:network_type flat --provider:physical_network physnet1
```

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 --tenant-id services 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-subnet 10.10.10.0/24 --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 --tenant-id services 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
```

c. For Fedora18 Image:

```
wget http://mattdm.fedorapeople.org/cloud-images/Fedora18-Cloud-x86_64-latest.qcow2
glance add name="Fedora18-x86_64" is_public=true container_format=ovf
disk_format=qcow2 < Fedora18-Cloud-x86_64-latest.qcow2
```

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:

```
nova keypair-add your-key-name > your-key-name.pem
```

Step 8 Boot an Instance (Precise image example):

```
quantum net-list |awk '/net10/ {print $2}'
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. Fedora18 Image Example

```
nova boot --image Fedora18-x86_64 --flavor m1.tiny --key_name <key_name> --nic
net-id=<quantum-net10-id> <your_instance_name>
```

c. 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 get the qrouter namespace, 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:

```
ip netns list | grep qrouter
ip netns exec <qrouter-namespace> ip addr list
ip netns exec <qrouter-namespace> ping <fixed-ip-of-instance>
ip netns exec <qrouter-namespace> 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 port-list |awk '/<fixed-ip-of-instance>/ {print $2}'
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 3548) 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 3548) Reference Bill of Materials

Product	Description	Quantity
N3K-C3548P-FA-L3A	Nexus 3548+L3+Algo, Fwd Airflow (port side exhaust), AC	1
CAB-C13-C14-2M	Power Cord Jumper, C13-C14 Connectors, 2 Meter Length	2
GLC-T	1000BASE-T SFP	16
SFP-H10GB-CU1M	10GBASE-CU SFP+ Cable 1 Meter	16
SFP-H10GB-CU3M	10GBASE-CU SFP+ Cable 3 Meter	16
N3KUK9-503A1.1	NX-OS Release 5.0(3)A1(1)	1
DCNM-LAN-N3K-K9	DCNM for LAN Advanced Edt. for Nexus 3000	1
N2200-PAC-400W	N2K/N3K AC Power Supply, Std airflow (port side exhaust)	2
N3548-ALGK9	Nexus 3548 Algo Boost License	1
N3548-BAS1K9	Nexus 3548 Base License	1
N3548-LAN1K9	Nexus 3548 Layer 3 LAN Enterprise License	1
N3K-C3064-ACC-KIT	Nexus 3064PQ Accessory Kit	1
NXA-FAN-30CFM-F	Nexus 2K/3K Single Fan, std airflow (port side exhaust)	4

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