Contents

Overview

Building OpenStack with the Cisco Edition will enable you to deploy a multi-node OpenStack system with Quantum enabled network services management. The system will also enable a simple monitoring function based on Nagios, Collectd, and Graphite.

In the Cisco OpenStack Edition (COE), an initial build server outside of the OpenStack cluster is used to manage and automate the OpenStack software deployment. This build server primarily functions as a Puppet server for software deployment and configuration management onto the OpenStack cluster, as well as a Cobbler installation server to manage the PXE boot used for rapid bootstrapping of the OpenStack cluster.

Once the build server is installed and configured, it is used as an out-of-band automation and management workstation to bring up, control, and reconfigure (if later needed) the nodes of the OpenStack cluster. It also functions as a monitoring server to collect statistics about the health and performance of the OpenStack cluster, as well as to monitor the availability of the machines and services which comprise the OpenStack cluster.

This current deployment supports:

- Single control server
- Multiple compute nodes
- Quantum managed network (OpenVirtual Switch GRE tunnel based environment)

The current model does not support deployment of:

- Swift
- Cinder
- Ceilometer
- Other OpenStack community incubated projects

The current system is deployable on Cisco UCS B-series and C-series hardware along with Nexus "Top-of-Rack" class L2/L3 network switches, though using the deployment with other compute and network platforms is possible. The system leverages the Ubuntu 12.04 LTS ("Precise") Linux operating system with KVM as its base hypervisor platform and for the build/monitoring node, RedHat/CentOS/SL support with

KVM for the hypervisor function is expected in a future release. There are no known limitations to virtualized machines running in the OpenStack environement as long as they can run in a KVM environment. The system does not currently support deployment of multiple hypervisors via Puppet.

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

Building the environment

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.

COE Folsom requires that you have two physically or logically (VLAN) separated IP networks. You must have an external router or layer-3 switch that provides connectivity between these two networks. This is required in order to support Quantum and the Cloud-Init metadata server for Provider Router based deployments (similar to FlatDHCP + Floating IP support in the nova-network model): http://docs.openstack.org/folsom/openstack-network/admin/content/adv_cfg_l3_agent_metadata.html

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 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.

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.

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

```
http_proxy=http://your-proxy.address.com:80/
https_proxy=https://your-https-proxy.address.com:443/
no_proxy=your-build-node-name,*yourbuild.domain.name,127.0.0.1,127.0.1.1,localhost
```

You have two choices for 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

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/folsom-manifests/multi-node/install_os_puppet |
```

With a proxy, use:

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

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

Model 2: Run the Commands Manually

All should now install any pending security updates:

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

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

Get the Cisco Edition example manifests. Under the folsom-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/folsom-manifests ~/cisco-folsom-manifests/
cd ~/cisco-folsom-manifests
git checkout -q 2012.2.3
```

With a proxy:

```
https_proxy=http://proxy.example.com:80 git clone https://github.com/CiscoSystems/folsom-manifests
cd ~/cisco-folsom-manifests
https_proxy=http://proxy.example.com:80 git checkout -q 2012.2.3
```

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

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

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

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

Then get the Cisco Edition 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
```

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 puppetmaster 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.

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
```

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

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

When the puppet apply command runs, the puppet 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:

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

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

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
```

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
#
```

And now, you should be able to use cobbler to build your controller:

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

NOTE: 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

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.

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
```

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.

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

1. Create quantum public network.

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

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 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 --allocation-pool start=192.168.221.10,end=192.168.221.250 public 192.168.22
```

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 --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.

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-uuid>
```

5. Connect the virtual router to your external network:

```
quantum router-gateway-set router1 <public-net-id>
```

You must create a static route on your physical router/layer-3 switch to the subnet that you define for your Tenants (quantum subnet-create command). The next-hop for the static route is the quantum router public interface. Use the following command to find the public address of your Quantum router:

```
quantum port-list -- --device_id <router-id> --device_owner network:router_gateway
```

5a. Continuing with our example, here is the required static route that MUST be configured on the external physical router/layer-3 switch. If you used the before mentioned "--allocation-pool" flag then the first address in the pool will be used for the qrouter address. So, in that example the next-hop entry for the example below would be "192.168.221.10":

```
ip route 10.10.10.0 255.255.255.0 192.168.221.3
```

6. Download an image and add it to Glance:

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-s
```

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
```

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>
```

1. Example> nova keypair-add --pub_key id_rsa.pub dkp

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-net1
```

Cirros Image Example

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

Verify that your instance has spawned successfully:

```
nova show <your_instance_name>
```

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>

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>
```

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
```

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

Using Scripts to Setup a Test Quantum Network and Launch Instances

A "quick" way to get a similar environment to the one listed above running is to use the scripted quantum tests.

Clone the test repository:

```
git clone https://github.com/CiscoSystems/quantum-l3-test
```

Then follow the instructions in the README.md or follow these steps:

Assumptions
There are a few assumptions being made in this script:

- The script creates keys and places them in the /root/.ssh/ path therefore we are assuming you are running this script as root. If you are not running as 'root' then you need to change the path in the 'create_vm' file for the ssh-keygen line.
- If you have existing files in the /root/.ssh/ path then the script will see them and prompt you to overwrite. If you do not select to overwrite the existing keys then you may encounter a "permissions denied" error when SSH'ing into the test instance. If this happens then ensure that the key being referenced during the nova keypair-add step in the 'create_vm' script is correct.

Instructions
1. cd into the quantum-l3-test directory you just cloned:

```
cd quantum-l3-test
```

2) Run: ./create_vm this will run net_setup on its own

```
./create_vm
```

3) You will be prompted to enter your specific network values for the public/private networks as well as altering the default path where the Ubuntu Precise image is downloaded from (i.e. your own local mirror).

4) You should be able to log into the instance:

```
ssh ubuntu@{fixed or floating ip of instance}
```

To reset the Quantum settings created by the script and relaunch the test VM from scratch:

```
./reset
```

```
./create_vm
```

Congratulations

Congratulations! You have now completed a basic setup and should have a functional environment.

Next Steps

System and service health monitoring of your OpenStack cluster is set up as part of the puppet deployment process. Once you have created your first VM, you should start seeing OpenStack status information show up in the health monitoring. To view Nagios health monitoring, log into the web page at:

http://ip-of-your-build-node/nagios3/

using username ***admin*** and password ***Cisco123***.

To view Graphite statistics, access the web page at:

http://ip-of-your-build-node:8190/

You may also wish to read the OpenStack operations documents on the OpenStack documentation site. Many users find the OpenStack Operations Guide published in March of 2013 to be particularly helpful.