Contents

Overview

The OpenStack Grizzly Release All-In-One (AIO) deployment builds off of the Cisco OpenStack Installer (COI) Multinode instructions. The primary difference is how the Puppet site.pp manifest is defined and, of course, the number of OpenStack nodes. An All-In-One node is an OpenStack deployment that has all relevant roles installed on a single node (Control/Compute/Storage/Network).

This document will cover the deployment of several AIO scenarios to include:

- Model 1: All-In-One node using the local file system for Glance (image) storage and using the Provider Router with Private Networks model for tenant network access (FlatDHCP + FloatingIPs using Quantum Router)
- Model 2: All-In-One node using Ceph RADOS Block Devices (RBD) for Glance (image) storage and the Provider Router with Private Networks model for tenant network access

- Model 3: All-In-One node using either storage model from above with the Provider Network Extensions model (VLANs trunked into Compute node from ToR switch)
- Model 4: All-In-One node using either storage model from above with the FlatDHCP Networking model (FlatDHCP, no FloatingIPs, no Quantum Router)

Diagrams

Figure 1 illustrates the topology used in Models 1-2

Figure 1: AIO Provider Router with Private Networks Diagram

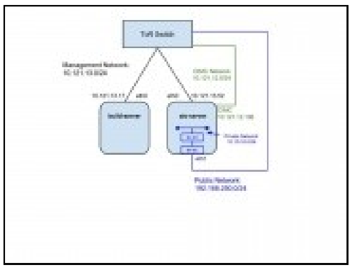


Figure 2 illustrates the topology used in Model 3

Figure 2: AIO Provider Network Extensions Diagram

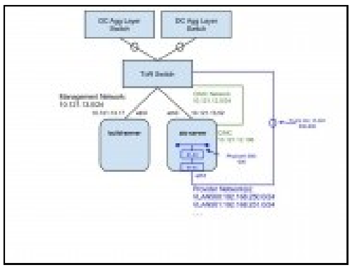
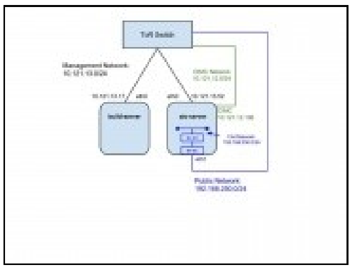


Figure 3 illustrates the topology used in Model 4

Figure 3: AIO FlatDHCP Networking Diagram



.

Building the environment

This section describes the process for deploying OpenStack with the Cisco OpenStack Installer in an All-In-One node configuration. In this document a build server will be deployed and used as the launching point for the AIO node.

Assumptions

Although other configurations are supported, the following instructions target an environment with a build node and an AIO node that is operating the OpenStack controller, compute, storage (Glance/Ceph) and network roles. 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 Cisco OpenStack Installer 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).

One network is used to provide connectivity for OpenStack API endpoints, Open vSwitch (OVS) GRE endpoints (especially important if multiple compute nodes are added to the AIO deployment), 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 OpenStack with COI, 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 though validation testing is done only using physical hardware.

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. 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 -i
```

NOTE: If you have proxies, or your AIO node does 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, AIO) 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 AIO node 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.

Method 1: Run the Script

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

```
curl -s -k -B https://raw.github.com/CiscoSystems/grizzly-manifests/g.3/install_os_puppet | /bin/b
```

With a proxy, use:

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

Note that the commands above install the g.3 release version of the top-level manifests. You can now jump to "Customizing your build server". Otherwise, follow along with the steps below.

Method 2: Run the Commands Manually

Install updates and other packages needed for this setup:

```
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 COI 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 even though our deployment is an AIO. The multi-node branch has an AIO section in the manifest:

```
git clone https://github.com/CiscoSystems/grizzly-manifests ~/cisco-grizzly-manifests/
cd ~/cisco-grizzly-manifests
git checkout -q g.3
```

With a proxy:

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

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

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

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

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

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

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

With a proxy:

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

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.py
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. **See the sections below for information on how the site.pp was modified for each model discussed in this document.**

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

Read each example deployment section below on how the site.pp file was modified for each scenario. Once you have modified the site.pp file to match your environment, continue on with the steps in "Applying the site.pp Manifest" section.

Example AIO site.pp Settings Per Model

You can find example site.pp files for each of the four previously mentioned models by browsing here:

- Model 1: https://github.com/shmcfarl/docwiki-manifests/blob/master/aio-filebackend.site.pp
- Model 2: https://github.com/shmcfarl/docwiki-manifests/blob/master/aio.cephbackend.site.pp
- Model 3: https://github.com/shmcfarl/docwiki-manifests/blob/master/aio.filebackend.provider.site.pp
- Model 4: https://github.com/shmcfarl/docwiki-manifests/blob/master/aio.cephbackend.flat.site.pp

Model 1: AIO with Filesystem as Backend to Glance using Provider Router with Private Networks

Figure 1 from the diagrams section shows the topology of Model 1. In the example site.pp file for Model 1 there are a few things that need to be highlighted that set this example apart from the Ceph/RBD backend example. Not all lines of the site.pp are discussed in this document as the site.pp file is internally documented but there are a few settings that need to be called out so you understand what defaults have been changed in the file:

- In all of the site.pp examples listed above, no proxy configuration is set. Pay close attention to the instructions in the document if you do use a proxy.
- Example build-server IP address information (aligns with previously shown diagram):

```
$cobbler_node_ip        = '10.121.13.17'
$node_subnet            = '10.121.13.0'
$node_netmask           = '255.255.255.0'
$node_gateway           = '10.121.13.1'
```

- OpenStack variables for the AIO node (known in site.pp file as the 'controller'). Ensure that the **$controller_node_address** value and the **$controller_hostname** values are consistent throughout the site.pp file:

```
$controller_node_address       = '10.121.13.52'
$controller_node_network       = '10.121.13.0'
$controller_hostname           = 'aio-server'
$db_allowed_network            = '10.121.13.%'
```

- The COI default site.pp (site.pp.example) uses the filesystem as the backend for Glance. In Model 1, this is left at its default:

```
$glance_backend      = 'file'
```

- Here is the example cobbler node definition that includes the eth0 MAC address for the PXE boot process, the IP address being assigned to the aio-server on eth0 and the CIMC IP address, user, password and powertype that will be used to remotely power-on/reboot the node so that the PXE boot process can begin:

```
  cobbler_node { 'aio-server':
   mac            => '00:10:18:CF:A6:A0',
   ip             => '10.121.13.52',
   power_address  => '10.121.12.106',
   power_user     => 'admin',
   power_password => 'password',
   power_type     => 'ipmitool',
  }
```

- Ensure the node definition has the correct hostname and using the '**allinone**' class:

```
node 'aio-server' inherits os_base {
  class { 'allinone': }
```

Model 2: AIO using Ceph RBD for Glance backend using Provider Router with Private Networks

Figure 1 from the diagrams section shows the topology of Model 2. In the example site.pp file for Model 2 the only changes that are shown below are those specific to Ceph with RBD as a backend for Glance. All of the IP addressing, node definitions and other variables are the same as in Model 1. As previously noted, the examples in this document are not using a proxy, so be mindful of the documentation within the site.pp file on setting specific values if a proxy is used. Also, this document gives no explanation of the specific Ceph settings shown below. If you are not familiar with Ceph, RBD, Cinder and other storage details, please read the Ceph COI Installation document and the Ceph documentation:

- Change the default **$glance_backend** value to 'rbd'

```
$glance_backend      = 'rbd'
```

- Set the RBD-backed Glance values:

```
$glance_ceph_enabled = true
$glance_ceph_user    = 'admin'
$glance_ceph_pool    = 'images'
```

- Set the **$ceph_combo** value so that the Monitor (MON) software and Object Storage Device (OSD) are on the same node:

```
$ceph_combo = true
```

- Set the various storage configuration values for Cinder/Ceph:

```
$cinder_controller_enabled     = true
$cinder_compute_enabled        = true
$cinder_storage_driver         = 'rbd'
$cinder_ceph_enabled           = true
```

- Run Ceph mon0 on the AIO node:

```
  if !empty($::ceph_admin_key) {
    @@ceph::key { 'admin':
      secret       => $::ceph_admin_key,
      keyring_path => '/etc/ceph/keyring',
    }
  }
  class {'ceph_mon': id => 0 }
```

- Set the Ceph OSD disk device and address values. In this example, there is a second physical drive in the UCS that is on /dev/sdb that will be used for the OSD disk device:

```
class { 'ceph::osd':
   public_address  => '10.121.13.52',
   cluster_address => '10.121.13.52',
 }

  ceph::osd::device { '/dev/sdb': }

}
```

- Configure Ceph values:

```
$ceph_auth_type          = 'cephx'
$ceph_monitor_fsid       = 'e80afa94-a64c-486c-9e34-d55e85f26406'
$ceph_monitor_secret     = 'AQAJzNxR+PNRIRAA7yUp9hJJdWZ3PVz242Xjiw=='
$ceph_monitor_port       = '6789'
$ceph_monitor_address    = $::ipaddress
$ceph_cluster_network    = '10.121.13.0/24'
$ceph_public_network     = '10.121.13.0/24'
$ceph_release            = 'cuttlefish'
$cinder_rbd_user         = 'admin'
$cinder_rbd_pool         = 'volumes'
$cinder_rbd_secret_uuid = 'e80afa94-a64c-486c-9e34-d55e85f26406'
```

- Set the global path for puppet-ceph. If you are using a proxy then uncomment the "environment" line as well:

```
Exec {
  path        => '/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin',
#  environment => "https_proxy=$::proxy",
}
```

Model 3: AIO using the Provider Network Extensions Model (VLANs)

Figure 2 from the diagrams section shows the topology of Model 3. In the example site.pp file for Model 3 there are specific Quantum/OVS settings that are needed to successfully deploy a Provider Network Extensions model using VLANs with the COI. As previously mentioned, all of the IP addressing, MAC, node specifics and storage settings (file as a Glance backend in this example) are the same as the previous models and not discussed in this section:

- Set the **$ovs_vlan_ranges** to match that of the trunked VLANs configured on the Top of Rack (ToR)/Access layer switch. In this case VLANs 500-600:

```
$ovs_vlan_ranges = 'physnet1:500:600'
```

- Set the OVS bridge-to-physical interface uplink (eth1 in this example):

```
$ovs_bridge_uplinks = ['br-ex:eth1']
```

- Set the OVS network name-to-external bridge mapping:

```
$ovs_bridge_mappings = ['physnet1:br-ex']
```

- Set the network type to 'VLAN' instead of 'GRE' when using VLAN Provider Networks:

```
$tenant_network_type = 'vlan'
```

Model 4: AIO using FlatDHCP Networking

Figure 3 from the diagrams section shows the topology of Model 4. In the example site.pp file for Model 4 there are specific Quantum/OVS settings that are needed to successfully deploy a basic FlatDHCP networking setup with the COI and the AIO node. As previously mentioned, all of the IP addressing, MAC, node specifics and storage settings are the same as the previous models and not discussed in this section:

- Set the **$ovs_vlan_ranges** to use a physical network name (no VLANs used):

```
$ovs_vlan_ranges = 'physnet1'
```

- Set the OVS bridge-to-physical interface uplink (eth1 in this example):

```
$ovs_bridge_uplinks = ['br-ex:eth1']
```

- Set the OVS network name-to-external bridge mapping:

```
$ovs_bridge_mappings = ['physnet1:br-ex']
```

- Set the network type to 'GRE' (no VLANs are being trunked into the AIO node in this example):

```
$tenant_network_type = 'gre'
```

Applying the site.pp Manifest

Once the site.pp file is as you want it, 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
  aio-server
```

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

```
/etc/puppet/manifests/clean_node.sh {node_name}
# Example:
/etc/puppet/manifests/clean_node.sh aio-server
```

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 AIO node as cobbler completes the automated install of Ubuntu. Once the installation finishes, the AIO node will reboot and then will run puppet after it boots up. Puppet will pull and apply the AIO 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 AIO node. /var/log/syslog on the AIO node will display the progress of the puppet configuration run.

NOTE: It may take more than one puppet run for the AIO 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 AIO configuration has converged completely to the configuration defined in puppet. **If you use the Ceph/RBD backend configuration then you will need to re-run the puppet agent a minimum of three times in order for the drives and relevant Ceph configuration to be completed. You will see errors that differ between each run until the Ceph configuration is completed.** You can re-run the puppet agent on the AIO node manually by using the following command:

```
puppet agent -t
```

Once the OpenStack AIO node has been built using cobbler, run puppet agent on the build node:

```
puppet agent -t
```

This 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 AIO node is 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/).

Deploy Your First VM on Models 1 & 2

The following deployment steps should be used after completing clean puppet runs on the OpenStack AIO node and restarting quantum-server and quantum-plugin-openvswitch-agent services. As mentioned in the previous section, you will need to source the openrc file in the root's home directory:

```
root@aio-server:~# source openrc
```

Define the Network

1. Create a Quantum public network.

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

2. 192.168.250.0/24 is being used as the subnet example on the external network. **Note:** The eth settings on the AIO 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 192.168.250.0/24
```

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 192.168.250.1) and the first addresses for floating-IPs will begin at .10:

```
quantum subnet-create --name public-subnet --allocation-pool start=192.168.250.10,end=192.168.250.
```

3. Create the internal (data) network (e.g. "net10") used for Tenants and create a subnet (e.g. "net10-subnet") and associate it with the network (e.g. "net10"). Create additional networks and associated subnets as needed. In the example below, specific DNS servers are being assigned that will be used by the instances:

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

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
```

5. Connect the virtual router to your external network:

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

Download and Add the Images

1. 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
```

Create the Keypair

- Create an SSH keypair and add the public key to Nova. Note: leave the passphrase empty when creating the keypair:

```
ssh-keygen
cd /root/.ssh/
nova keypair-add --pub_key id_rsa.pub <key_name>
# Example
nova keypair-add --pub_key id_rsa.pub ctrl-key
```

Boot an Instance

1. Boot an Instance (Precise image example). Run the "quantum net-list" command to get a list of networks. Use the ID from the net-list output in the **--nic net-id=** field:

```
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. **Note:** The first time an instance is launched on the system it can take a bit longer to boot than subsequent launches of instances:

```
nova show <your_instance_name>
```

2. Verify connectivity to the instance from the AIO node. Since namespaces are being used in this model, you will need to run the commands from the context of the qrouter using the "ip netns exec qrouter" syntax. List the qrouter to get its router-id, connect to the qrouter and get a list of its addresses, ping the instance from the qrouter and then 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>

3. 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>
```

4. Enable Ping and SSH to Instances:

```
quantum security-group-rule-create --protocol icmp --direction ingress default
quantum security-group-rule-create --protocol tcp --port-range-min 22 --port-range-max 22 --direct
```

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

Deploy Your First VM on Model 3

- Source the openrc file in the /root/ directory:

```
root@aio-server:~# source openrc
```

Define the Network

1. Create a Quantum provider network using VLANs. In this example the **segmentation_id 500** maps to VLAN 500 that is being trunked to the host:

```
quantum net-create vlan500 --provider:network_type vlan --provider:physical_network physnet1 --prc
```

2. Again, 192.168.250.0/24 is being used as the subnet on our external network. Given that the external network that is being trunk into the AIO node is used on the ToR switch and any upstream Data Center Aggregation Layer switches, you need to ensure that the subnet being assigned does not have addresses that conflict with any addresses being used by the tenant(s). A good example of this is if the upstream L3 switches use HSRP/GLBP/VRRP and use low-order IP addresses such as .2 and .3. The default subnet address assignments used by Quantum for this subnet such as DHCP agent or the instances themselves 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 192.168.250.1) and the first addresses for floating-IPs will begin at .10: Also, the port on the ToR switch connecting to the ethernet port on the AIO node that is being used for tenant VLAN connectivity will need to be in trunk mode and allowing the VLANs (i.e. VLAN 500-600) on that trunk port. A Nexus switch example is given below.

```
quantum subnet-create --name subnet500 --allocation-pool start=192.168.250.10,end=192.168.250.254
```

Here is an example Nexus ToR switch configuration that can be used with this network scenario (refer to this topology as a reference):

```
vlan 500
  name OS-AIO-VLAN-500

interface Ethernet1/9
  description to aio-server eth0
  switchport mode trunk
  switchport trunk allowed vlan 13,500-600
  speed 1000

interface Ethernet1/10
  description to aio-server eth1
  switchport mode trunk
  switchport trunk allowed vlan 500-600
  speed 1000
```

Download and Add the Images

1. 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
```

Create the Keypair

- Create an SSH keypair and add the public key to Nova. Note: leave the passphrase empty when creating the keypair:

```
ssh-keygen
cd /root/.ssh/
nova keypair-add --pub_key id_rsa.pub <key_name>
# Example
nova keypair-add --pub_key id_rsa.pub ctrl-key
```

Boot an Instance

1. 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-vla
```

Cirros Image Example

```
nova boot --image cirros-x86_64 --flavor m1.tiny --key_name <key_name> --nic net-id=<quantum-vlan5
```

Verify that your instance has spawned successfully:

```
nova show <your_instance_name>
```

2. Enable Ping and SSH to Instances:

```
quantum security-group-rule-create --protocol icmp --direction ingress default
quantum security-group-rule-create --protocol tcp --port-range-min 22 --port-range-max 22 --direct
```

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

Deploy Your First VM on Model 4

- Source the openrc file in the /root/ directory:

```
root@aio-server:~# source openrc
```

Define the Network

1. Create a Quantum flat network:

```
quantum net-create public --provider:network_type flat --provider:physical_network physnet1  --sha
```

2. Create a Quantum subnet:

```
quantum subnet-create --name public-subnet --allocation-pool start=192.168.250.10,end=192.168.250.
```

Download and Add the Images

1. 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
```

Create the Keypair

- Create an SSH keypair and add the public key to Nova. Note: leave the passphrase empty when creating the keypair:

```
ssh-keygen
cd /root/.ssh/
nova keypair-add --pub_key id_rsa.pub <key_name>
# Example
nova keypair-add --pub_key id_rsa.pub ctrl-key
```

Boot an Instance

1. 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-pub
```

Cirros Image Example

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

Verify that your instance has spawned successfully:

```
nova show <your_instance_name>
```

2. Enable Ping and SSH to Instances:

```
quantum security-group-rule-create --protocol icmp --direction ingress default
quantum security-group-rule-create --protocol tcp --port-range-min 22 --port-range-max 22 --direct
```

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

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.

Need help?

Cisco does not currently provide TAC support for Cisco OpenStack Installer. However, you may contact our developers at openstack-support@cisco.com for best-effort technical help. For general OpenStack questions, you may also wish to peruse http://ask.openstack.org and ask your question there to solicit a wider audience.

Authors

Shannon McFarland (@eyepv6) - Principal Engineer