Instructions from:

https://github.com/danehans/puppet-openstack/blob/essex-ha repository

Contents

The OpenStack High Availability Modules:

Introduction

The Openstack High Availability (HA) Puppet Modules are a flexible Puppet implementation capable of configuring Openstack and additional services for providing high-availability mechanisms. A 'Puppet Module' is a collection of related content that can be used to model the configuration of a discrete service.

The OpenStack HA solution provides active/active redundancy for Controller Nodes, Compute Nodes, Swift Proxy Nodes and Load-Balancer Nodes. Compute Nodes employ the well-known multi-host HA networking option to eliminate a nova-network single point of failure.

The modules currently support the OpenStack Essex release, but Folsom is expected to be supported 30-days after the Grizzly Design Summit:

- Nova (Compute Service)
- Glance (Image Service)
- Swift (Object Storage Service)
- Keystone (Authentication Service)
- Horizon (OpenStack Dashboard Web User Interface)

These modules are based on the administrative guides for OpenStack Compute and Object Store

Dependencies:

Puppet:

- Puppet 2.7.11 or greater
- Facter 1.6.5 or greater (versions that support the osfamily fact)

Operating System Platforms:

These modules have been fully tested on Ubuntu 12.04 LTS (Precise).

Networking:

Each of the servers running OpenStack services should have a minimum of 2 networks, and preferably 3 networks. The networks can be physically or virtually (VLAN) separated. In addition to the 2 OpenStack networks, it is recommended to have an ILO/CIMC network to fully leverage the remote management capabilities of the Cobbler Module. Additionally, puppet-networking models OpenStack network configurations.

The following provides a brief explanation of the OpenStack Module networking requirements.

- OpenStack Management Network

    - This network is used to perform management functions against the node, Puppet Master <> Agent is an example.
    - An IP address for each node is required for this network.
    - This network typically employs private (RFC 1918) IP addressing.
- Nova Public/API Network
    - This network is used for assigning Floating IP addresses to instances, for communicating outside of the OpenStack cloud, etc..
    - An IP address for the node is required for this network.
    - (Optional) This network can be collapsed with the OpenStack Management Network.
    - This network typically employs publicly routable IP addressing.
- Instance Network
    - This network is used for providing connectivity to OpenStack Instances using either the Flat or VLAN Nova Networking Manager.
    - An IP address for the node is not necessary, as Nova automatically creates a bridge interface with an IP address.
    - This network typically employs private (RFC 1918) IP addressing.

Storage Volumes:

Every Compute Node is configured to host the nova-volume service to provide persistent storage to instances through iSCSI. The volume-group name is 'nova-volumes' and should not be changed.

Node Types:

The OpenStack HA solution consists of 6 Nodes Types:

- Build Node
    - Minimum Quantity: 1
    - Runs Puppet Master, Cobbler, and other management services.
    - Provides capabilities for managing OpenStack environments at scale.

- Load Balancer Node
    - Minimum Quantity: 2
    - Runs HAProxy and Keeplived.
    - Provides monitoring and fail-over for API endpoints and between load-balancer nodes.

- Controller Node
    - Minimum Quantity: 3
    - Runs MySQL Galera, Keystone, Glance, Nova, Horizon, and RabbitMQ.
    - Provides control plane functionality for managing the OpenStack Nova environment.

- Compute Node
    - Minimum Quantity: 1 (recommend having 2 or more to demonstrate nova-scheduler across multiple nodes)
    - Runs the following Nova services: api, compute, network, and volume.
    - Provides necessary infrastructure services to Nova Instances.

- Swift Proxy Node
    - Minimum Quantity: 2
    - Runs swift-proxy, memcached, and keystone-client.
    - Authenticates users against Keystone and acts as a translation layer between clients and storage.

- Swift Storage Node
  - ♦ Minimum Quantity: 3
  - ♦ Runs Swift account/container/object services. XFS is used as the filesystem.
  - ♦ Controls storage of the account databases, container databases, and the stored objects.

Installation

Installation Order

The OpenStack Nodes are required to be deployed in a very specific order. For the time being, you need to perform multiple puppet runs for most Nodes to deploy properly. The following is the order in which the nodes should be deployed. Preface commands with **sudo** if you are not the root user:

- **HAProxy Nodes**: Make sure the haproxy/keepalived services are running before proceeding to the next node type.
- **Swift Storage Nodes**: If you are rebuilding Swift Storage Nodes, the hard disks should be zero'ed out. Use the clean-disk script from the Cisco repo or use the following command on each storage node before starting the rebuild:

```
for i in  b c d e f <add/subtract drive letters as needed>
do
dd bs=1M count=1000 if=/dev/zero of=/dev/sd$i
done
```

- **Swift Proxy Node #1**: Make sure the ring is functional before adding the 2nd Proxy.
- **Swift Proxy Node 2**: Make sure the ring is functional before proceeding.
- **Controller Nodes 1-3**: You must ensure that the HAproxy Virtual IP address for the Controller cluster is working or your puppet run will fail. Deploy Controllers one at a time starting with Controller 1.
- **Compute Nodes**: Start off with just 1 or 2 nodes before deploying a large number.
- Test to make sure environment is functional.

Install the Build Node

A Build Node (Puppet Master, Cobbler, etc.) is required for deploying the OpenStack HA environment. Follow the Cisco Build Node Deployment Guide for step-by-step instructions of this process.

- In addition to the steps identified in the Cisco Build Node Deployment Guide, Rake and Git should also be installed on the Puppet Master:

```
apt-get install rake git -y
```

Install the OpenStack HA Modules

The OpenStack HA modules should be installed into the module path of your Build Node. **Note:** If they exist, remove any existing modules from the modulepath.

Modulepath:

- Opensource Puppet - /etc/puppet/modules
- Install the latest version of the modules from git:

```
cd <module_path>
```

```
git clone git://github.com/danehans/puppet-openstack openstack

cd openstack

git checkout -t -b essex-ha remotes/origin/essex-ha

rake modules:clone
```

- Copy the example OpenStack HA manifests to your manifests directory. **Note:** If they exist, remove any duplicate manifests from the manifests directory (/etc/puppet/manifests).

```
cp <module_path>/openstack/examples/site.pp /etc/puppet/manifests/site.pp
cp <module_path>/openstack/examples/haproxy-nodes.pp /etc/puppet/manifests/haproxy-nodes.pp
cp <module_path>/openstack/examples/cobbler-node.pp /etc/puppet/manifests/cobbler-node.pp
cp <module_path>/openstack/examples/swift-nodes.pp /etc/puppet/manifests/swift-nodes.pp
cp <module_path>/openstack/examples/clean-disk.pp /etc/puppet/manifests/clean-disk.pp
```

- Edit the manifests according to your deployment needs. At a minimum, the following should be changed:
    - IP addressing
    - Node name definitions
    - DNS naming
    - user/password information
    - (Optional) interface definitions
    - (Optional) additional Compute Node and Swift Storage Node definitions
    - (Optional) additional Swift Storage Node hard drive definitions
- The proceeding sections will detail the example manifests and configuration options.

Overview of Key Modules

puppet-openstack module

The 'puppet-openstack' module was written for users interested in deploying and managing a production-grade, highly-available OpenStack deployment. It provides a simple and flexible means of deploying OpenStack, and is based on best practices shaped by companies that contributed to the design of these modules.

puppet-cobbler module

The 'puppet-cobbler' module is used to provide several key tasks such as, bare-metal OS provisioning, ILO management of servers, etc..

puppet-swift module

The 'puppet-swift' module manages all configuration aspects of Swift Proxy and Swift Storage Nodes. The module relies on underlying technologies/modules to deliver object storage functionality to your OpenStack environment.

puppet-haproxy module

The 'puppet-haproxy' module provides load-balancing services for API endpoints.

Overview of Example Manifests

cobbler-node manifest

- Shared Variables
- $cobbler_node_ip: Is the IP address assigned to the Build Node and referenced throughout the manifest.

cobbler class

This class installs/configures Cobbler and the PXE boot install engine.

- Usage Example:

A cobbler class is configured as follows:

```
class { cobbler:
  node_subnet => '192.168.220.0',
  node_netmask => '255.255.255.0',
  node_gateway => '192.168.220.1',
  node_dns => "${cobbler_node_ip}",
  ip => "${cobbler_node_ip}",
  dns_service => 'dnsmasq',
  dhcp_service => 'dnsmasq',
  dhcp_ip_low => '192.168.220.240',
  dhcp_ip_high => '192.168.220.250',
  domain_name => 'dmz-pod2.lab',
  proxy => "http://${cobbler_node_ip}:3142/",
  password_crypted => '$6$UfgWxrIv$k4KfzAEMqMg.fppmSOTd0usI4j6gfjs0962.JXsoJRWa5wMz8yQk4SfInn4.WZ3
}
```

Note: The encrypted password (password_crypted) is ubuntu. For more information on the parameters, check out the inline documentation in the manifest:

```
module_path/cobbler/manifests/init.pp
```

cobbler::ubuntu

This class manages the Ubuntu ISO used to PXE boot servers.

- Usage Example:

This class will load the Ubuntu Precise x86_64 server ISO into Cobbler:

```
cobbler::ubuntu { "precise":}
```

For more information on the parameters, review the inline documentation within the manifest:

```
module_path/cobbler/manifests/ubuntu.pp
```

cobbler::node

This manifest installs a node into the cobbler system. Run puppet apply -v /etc/puppet/manifests/site.pp after adding nodes to cobbler::node. You can use the 'cobbler system list' command to verify that nodes have been properly added to Cobbler.

- Usage Example:

A cobbler::node is configured as follows:

```
cobbler::node { "control01":
  mac => "A4:4C:11:13:8B:D2",
  profile => "precise-x86_64-auto",
  domain => "yourdomain.com",
  preseed => "/etc/cobbler/preseeds/cisco-preseed",
  power_address => "192.168.220.2",
  power_type => "ipmitool",
  power_user => "admin",
  power_password => "password",
 }
```

- Most of the parameters are self explanatory, however here are a few parameters that require additional explanation:
- **[power_address]** IP address of CIMC interface of server.
- **[power_type]** supported options include 'UCS' for UCS B-Series and 'ipmitool' for UCS C-Series servers
- **[power_user]** and **[power_password]** username/password for a user account with permissions to manage the server.
- **[Power-ID]** (For UCS B-Series servers only) needs to map to the service profile name within UCS Manager.

For more information on the parameters, check out the inline documentation in the manifest:

```
module_path/cobbler/manifests/node.pp
```

site manifest

The site manifest provides the top-level configuration interface to the OpenStack HA environment. I will outline the example manifest so users can customize it as needed.

- The first module configured is apt. Apt provides helpful definitions for dealing with package management.
  - class { 'apt': } This module manages the initial configuration of apt.
  - [apt::ppa] Adds the Cisco Edition ppa repository using `add-apt-repository`.
  - [apt::pin] Pins packages from the Cisco Edition software repositories.
- Shared Variables: I will not address every shared variable, as many of them are self explanatory or include inline documentation. Here is a list of shared variables that can use additional explanation:
  - **[$multi_host]** Must be set to true. Establishes the Nova multi-host networking model
  - **[$rabbit_addresses]** List of Controller addresses used in nova.conf for RabbitMQ Mirrored Queueing
  - **[$memcached_servers]** List of Controller addresses in /etc/nova/nova.conf used for Nova Consoleauth redundancy
  - **[$swift_proxy_address]** This should be the Virtual IP (VIP) address of your Swift Proxy Nodes.
  - **[$mysql_puppet_password]** The password for the puppet user of the PuppetMaster database.
  - **[$horizon_secret_key]** Used by the secret key generator of the horizon.utils.secret_key Module.
  - **[$cluster_rabbit]** Keep setting as true. Enables/disables RabbitMQ clustering.
  - **[$rabbit_cluster_disk_nodes]** Used to specify Controller Nodes that will be used for RabbitMQ clustering.

- **Import Section:** imports other manifests from /etc/puppet/manifests/<manifest_name> into the site manifest.
- **Node Base:** Defines a base node that is imported into other node definitions, except the Build Node.

networking::interfaces class

This class manages the /etc/network/interfaces file for OpenStack HA Nodes.

- Usage Example:

This class will configure networking parameters for OpenStack Nodes. This particular example configures networking for a Controller Node that uses VLAN 221 for the Nova Instance (flat) network and collapses the public/management networks:

```
class { 'networking::interfaces':
  node_type           => controller,
  mgt_is_public       => true,
  vlan_networking     => true,
  vlan_interface      => "eth0",
  mgt_interface       => "eth0",
  mgt_ip              => "192.168.220.41",
  mgt_gateway         => "192.168.220.1",
  flat_vlan           => "221",
  flat_ip             => "10.0.0.251",
  dns_servers         => "192.168.220.254",
  dns_search          => "dmz-pod2.lab",
 }
```

For more information on the parameters, check out the inline documentation in the manifest:

module_path/networking/manifests/interfaces.pp

galera class

This class manages MySQL Galera on Controller Nodes. Galera is used to provide multi-master redundancy and scalability of the Controller Node MySQL databases.

- Usage Example:

This class will manage Galera on the 1st Controller in the Controller cluster. Make sure to uncomment **master_ip** after the 2nd Controller Node is deployed and the Galera cluster is sync'ed.

For the 2nd and 3rd Controllers, make sure the master_ip is $controller_node_primary:

```
class { 'galera' :
  cluster_name  => 'openstack',
  #master_ip    => $controller_node_secondary,
 }
```

You can verify if the database is sync'ed by using the following mysql command:

```
mysql> show status like 'wsrep%';
```

For more information on the parameters, check out the inline documentation in the manifest:

```
<module_path>/galera/manifests/init.pp
```

galera::haproxy class

This class adds a service user account to the database that allows HAProxy to monitor the database cluster health. **Note:** If you change the user account information here, you need to also change the **mysql-check user** definition in the the haproxy::config manifest.

- Usage Example:

```
class {'galera::haproxy': }
```

If needed, you can define account credentials for the service account. For more information on the parameters, check out the inline documentation in the manifest:

```
module_path/galera/manifests/haproxy.pp
```

openstack::controller

The openstack::controller class is intended to provide support for HA OpenStack deployments. The module focuses on the Nova, Keystone, and Glance services.

There are two manifests that manage the Openstack Nova HA deployment:

- **controller-** Deploys all of the Controller Node core, middleware and high-availability services.
- **compute-** Deploys Nova API, Compute, Network and Volume services.

The openstack::controller class deploys the following OpenStack services: Keystone, Horizon, Glance, Nova.

- Usage Example:

An Openstack Controller class is configured as follows:

```
class { 'openstack::controller':
  public_address          => $controller_node_public,
  virtual_address         => $controller_node_address,
  public_interface        => $public_interface,
  private_interface       => $private_interface,
  internal_address        => $internal_address,
  floating_range          => $floating_ip_range,
  fixed_range             => $fixed_network_range,
  multi_host              => $multi_host,
  network_manager         => 'nova.network.manager.FlatDHCPManager',
  verbose                 => $verbose,
  auto_assign_floating_ip => $auto_assign_floating_ip,
  admin_email             => $admin_email,
  admin_password          => $admin_password,
  keystone_host           => $controller_node_address,
  keystone_db_password    => $keystone_db_password,
  keystone_admin_token    => $keystone_admin_token,
  glance_db_password      => $glance_db_password,
  glance_user_password    => $glance_user_password,
  glance_on_swift         => $glance_on_swift,
  nova_db_password        => $nova_db_password,
  nova_user_password      => $nova_user_password,
  horizon_secret_key      => $horizon_secret_key,
  memcached_servers       => $memcached_servers,
  cache_server_ip         => $internal_address,
  rabbit_password         => $rabbit_password,
  rabbit_user             => $rabbit_user,
```

```
  cluster_rabbit          => $cluster_rabbit,
  cluster_disk_nodes      => $rabbit_cluster_disk_nodes,
  api_bind_address        => $internal_address,
  export_resources        => false,
 }
```

For more information on the parameters, check out the inline documentation in the manifest:

```
<module_path>/openstack/manifests/controller.pp
```

openstack::compute class

The Openstack compute class is used to manage Nova Compute Nodes. A typical Openstack HA installation would consist of at least 3 Controller Nodes and a large number of Compute Nodes (based on the amount of resources being virtualized)

The openstack::compute class deploys the following services: * nova - nova-compute (libvirt backend) - nova-network service (multi_host must be enabled) - nova-api service (multi_host must be enabled) - nova-volume

- Usage Example:

An OpenStack Compute class can be configured as follows:

```
class { 'openstack::compute':
  public_interface   => $public_interface,
  private_interface  => $private_interface,
  internal_address   => $internal_address,
  virtual_address    => $controller_node_address,
  libvirt_type       => 'kvm',
  fixed_range        => $fixed_network_range,
  network_manager    => 'nova.network.manager.FlatDHCPManager',
  multi_host         => $multi_host,
  nova_user_password => $nova_user_password,
  nova_db_password   => $nova_db_password,
  rabbit_password    => $rabbit_password,
  rabbit_user        => $rabbit_user,
  api_bind_address   => $internal_address,
  vncproxy_host      => $controller_node_address,
  vnc_enabled        => 'true',
  verbose            => $verbose,
  manage_volumes     => true,
  nova_volume        => 'nova-volumes',
 }
```

For more information on the parameters, check out the inline documentation in the manifest:

```
module_path/openstack/manifests/compute.pp
```

Customizing Manifests

So far, classes have been discussed as configuration interfaces used to deploy the OpenStack roles. This section explains how to apply these roles to actual nodes using a puppet site manifest.

The default file name for the site manifest is site.pp. This file should be contained in the puppetmaster's manifestdir:

- Cisco Edition- /etc/puppet/manifests/site.pp
- Open-source Puppet - /etc/puppet/manifests/site.pp
- Puppet Enterprise - /etc/puppetlabs/puppet/manifests/site.pp

Node blocks are used to map a node's certificate name to the classes that should be assigned to it.

Node blocks can match specific hosts:

```
node my_explicit_host {...}
```

Or they can use regular expression to match sets of hosts

```
node /my_similar_hosts/ {...}
```

Inside the site.pp file, Puppet resources declared within node blocks are applied to those specified nodes. Resources specified at top-scope are applied to all nodes.

Deploying HAProxy Load-Balancers

The servers that act as your load-balancers should be managed by Cobbler.
Make sure you your cobbler-node manifest is properly configured and you have added node definitions for your two load-balancers nodes. Here is an example:

```
cobbler::node { "slb01":
  mac => "A4:4C:11:13:44:93",
  profile => "precise-x86_64-auto",
  domain => "dmz-pod2.lab",
  preseed => "/etc/cobbler/preseeds/cisco-preseed",
  power_address => "192.168.220.8",
  power_type => "ipmitool",
  power_user => "admin",
  power_password => "password",
 }

cobbler::node { "slb02":
  mac => "A4:4C:11:13:BA:17",
  profile => "precise-x86_64-auto",
  domain => "dmz-pod2.lab",
  preseed => "/etc/cobbler/preseeds/cisco-preseed",
  power_address => "192.168.220.10",
  power_type => "ipmitool",
  power_user => "admin",
  power_password => "password",
 }
```

puppet apply the site.pp file to add the nodes to Cobbler:

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

Verify that the nodes have been added to Cobbler

```
cobbler system list
```

Next, edit the node definitions and network settings in /etc/puppet/manifests/haproxy-nodes.pp. Replace <SLB-01> and <SLB-02> with the hostname/certname of your load-balancer nodes.

```
vi /etc/puppet/manifests/haproxy-nodes.pp
```

```
node /<SLB-01>/

node /<SLB-02>/

class { 'networking::interfaces':
  node_type           => load-balancer,
  mgt_is_public       => true,
  mgt_interface       => "eth0",
  mgt_ip              => "192.168.220.62",
  mgt_gateway         => "192.168.220.1",
  dns_servers         => "192.168.220.254",
  dns_search          => "dmz-pod2.lab",
 }
```

Lastly, use Cobbler to power-on the HAProxy Nodes and begin the deployment process:

```
sudo cobbler system poweron --name=<name of HAProxy Node1 from cobbler system list command>

sudo cobbler system poweron --name=<name of HAProxy Node2 from cobbler system list command>
```

Deploying Swift

The servers that act as your Swift Proxies and Storage Nodes should be managed by Cobbler. Make sure you your cobbler-node manifest is properly configured and you have added node definitions for your Swift Nodes. Here is an example:

```
cobbler::node { "swiftproxy01":
  mac => "A4:4C:11:13:44:93",
  profile => "precise-x86_64-auto",
  domain => "dmz-pod2.lab",
  preseed => "/etc/cobbler/preseeds/cisco-preseed",
  power_address => "192.168.220.8",
  power_type => "ipmitool",
  power_user => "admin",
  power_password => "password",
 }

cobbler::node { "swift01":
  mac => "A4:4C:11:13:BA:17",
  profile => "precise-x86_64-auto",
  domain => "dmz-pod2.lab",
  preseed => "/etc/cobbler/preseeds/cisco-preseed",
  power_address => "192.168.220.10",
  power_type => "ipmitool",
  power_user => "admin",
  power_password => "password",
 }
```

Next, edit the node definitions and network settings in /etc/puppet/manifests/swift-nodes.pp. Replace existing node definitions with the hostname/certname of your Swift Storage and Proxy Nodes. The site.pp file should include the '**import swift-nodes**' statement.

```
vi /etc/puppet/manifests/swift-nodes.pp

node /swift01/

node /swift02/

node /swift03/
```

```
node /swiftproxy01/

class { 'networking::interfaces':
  node_type           => load-balancer,
  mgt_is_public       => true,
  mgt_interface       => "eth0",
  mgt_ip              => "192.168.220.62",
  mgt_gateway         => "192.168.220.1",
  dns_servers         => "192.168.220.254",
  dns_search          => "dmz-pod2.lab",
 }
```

Note: Do not define the 2nd Swift Proxy until the Storage Nodes and first proxy are deployed and the ring is established. Also, add additional Storage Node definitions as needed. You must use at least 3 Storage Nodes to create a Swift ring.

Run puppet apply on the site.pp file of the Puppet Master to add the Swift Storage Nodes to Cobbler:

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

Verify that the nodes have been added to Cobbler

```
cobbler system list
```

The Swift Nodes must be configured in the following order:

- First the storage nodes need to be configured. This creates the storage services (object, container, account) and exports all of the storage endpoints for the ring builder (Proxy Node) into storeconfigs. **Note:** It is expected that the account, object, and container replication service will fail to start during the initial puppet run. You are ready to move to the first Proxy Node when you have reached this state in your puppet runs for the Storage Nodes.
- Next, Swift Proxy 1 should be deployed. The ringbuilder (included in the Proxy Node) collects the storage endpoints and creates the ring database. It also creates an rsync server which is used to host the ring database. The resources are used to rsync the ring database from the Swift Proxy.
- If you like, you can verify the contents of the Swift storeconfigs in the Puppet Master database:

```
mysql

use puppet;

select * from resources;
```

- Next, the storage nodes should be run again so that they can rsync the ring databases.
- Add the 2nd Proxy Node.
- From a Proxy Node, verify the contents of the Swift rings:

```
swift-ring-builder /etc/swift/account.builder

swift-ring-builder /etc/swift/container.builder

swift-ring-builder /etc/swift/object.builder
```

The example configuration creates five storage devices on every node. Make sure to increase/decrease the following swift-nodes.pp definitions based on the number of hard disks in your Storage Nodes:

```
swift::storage::disk

@@ring_object_device

@@ring_container_device

@@ring_account_device
```

Next, use Cobbler to power-on the Storage Nodes and begin the deployment process:

```
sudo cobbler system poweron --name=<name of Swift Storage Node 1 from cobbler system list command>

sudo cobbler system poweron --name=<name of Swift Storage Node 2 from cobbler system list command>

sudo cobbler system poweron --name=<name of Swift Storage Node 3 from cobbler system list command>
```

Repeat this step for the Proxy Node 1 after the Storage Nodes have been deployed. Repeat for Proxy Node 2, when the ring is established between the 3 Storage Nodes and Proxy Node 1.

Note: After the Swift environment has been deployed properly, you will see the following errors on Storage Nodes of subsequent puppet runs:

```
err: Could not retrieve catalog from remote server: Error 400 on SERVER:
Exported resource Swift::Ringsync[account] cannot override local resource on node <node_name>
warning: Not using cache on failed catalog
err: Could not retrieve catalog; skipping run
```

Comment-out the Swift::Ringsync<<||>> definition under the Storage Nodes or Class that gets imported into your Storage Node definitions.

```
#Swift::Ringsync<<||>>
```

Deploying an OpenStack Nova HA Environment

The servers that act as your Nova Controllers and Compute Nodes should be managed by Cobbler. Make sure you your cobbler-node manifest is properly configured and you have added node definitions for your Controller and Compute Nodes. Here is an example:

```
cobbler::node { "control01":
  mac => "A4:4C:11:13:44:93",
  profile => "precise-x86_64-auto",
  domain => "dmz-pod2.lab",
  preseed => "/etc/cobbler/preseeds/cisco-preseed",
  power_address => "192.168.220.8",
  power_type => "ipmitool",
  power_user => "admin",
  power_password => "password",
 }

cobbler::node { "compute01":
  mac => "A4:4C:11:13:BA:17",
  profile => "precise-x86_64-auto",
  domain => "dmz-pod2.lab",
  preseed => "/etc/cobbler/preseeds/cisco-preseed",
  power_address => "192.168.220.10",
  power_type => "ipmitool",
  power_user => "admin",
  power_password => "password",
 }
```

puppet apply the site.pp file to add the nodes to Cobbler:

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

Verify that the nodes have been added to Cobbler

```
cobbler system list
```

Next, edit the node definitions and network settings in /etc/puppet/manifests/site.pp. Replace control01, control02, control03, compute01 with the hostname/certname of your Controller/Compute Nodes. **Note:** Since Controller Nodes need to be deployed in order of 1-3, we suggest you edit site.pp node name definitions one-by-one and perform puppet runs. The same applies to your Compute Node(s). Otherwise the nodes

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

node /<control01>/

node /<control02>/

node /<control03>/

node /<compute01>/

class { 'networking::interfaces':
  # This is a Controller Node Example
  node_type           => controller,
  mgt_is_public       => true,
  vlan_networking     => true,
  vlan_interface      => "eth0",
  mgt_interface       => "eth0",
  mgt_ip              => "192.168.220.41",
  mgt_gateway         => "192.168.220.1",
  flat_vlan           => "221",
  flat_ip             => "10.0.0.251",
  dns_servers         => "192.168.220.254",
  dns_search          => "dmz-pod2.lab",
}

class { 'networking::interfaces':
  # This is a Compute Node Example
  node_type           => compute,
  mgt_is_public       => true,
  vlan_networking     => true,
  vlan_interface      => "eth0",
  mgt_interface       => "eth0",
  mgt_ip              => "192.168.220.51",
  mgt_gateway         => "192.168.220.1",
  flat_vlan           => "221",
  dns_servers         => "192.168.220.254",
  dns_search          => "dmz-pod2.lab",
}
```

Note: Add additional Compute Node definitions as needed.

Lastly, use Cobbler to power-on Controller Node 1 and begin the deployment process:

```
sudo cobbler system poweron --name=<name of Controller Node 1 from cobbler system list command>
```

Repeat this step for the 2 other Controller Nodes after the first Controller Node has been deployed. Repeat for Compute Nodes, when the Controllers have been successfully deployed.

Keep in mind that the deployment ***MUST*** be performed in a very specific order (outlined above). You can either make all the necessary changes to your site manifests and keep particular nodes powered-off, or you can change site.pp node name definition one-by-one and perform puppet runs.

Verifying an OpenStack deployment

Once you have installed openstack using Puppet (and assuming you experience no errors), the next step is to verify the installation:

openstack::auth_file

The optionstack::auth_file class creates the file:

```
/root/openrc
```

Which stores environment variables that can be used for authentication of OpenStack command line utilities.

- Usage Example:

```
class { 'openstack::auth_file':
  admin_password       => 'my_admin_password',
  controller_node      => 'my_controller_node',
  keystone_admin_token => 'my_admin_token',
}
```

```
source /root/openrc
```

openstack::test_file

The optionstack::test_file class creates the file:

```
/tmp/test_nova.sh
```

Which is a script that tests the basic functionality of your OpenStack environment. Feel free to review the script to better understand the functionality it provides. Execute the script as follows:

```
. /tmp/test_nova.sh
```

- Usage Example:

```
class { 'openstack::test_file': }
```

Verification Process

Follow the proceeding steps if you prefer to manually verify your OpenStack HA implementation.

1. Verify your authentication information. **Note:** This assumes that the class openstack::auth_file had been applied to the node you are on.

   ```
   cat /root/openrc
   ```
2. Ensure that your authentication information is in the user's environment.

```
source /root/openrc
```

3. Verify Keystone is operational:

```
service keystone status
```

4. Ensure the Keystone Service Endpoints have been properly configured:

```
keystone endpoint-list
```

5. Verify glance is operational:

```
service glance-api status

service glance-registry status
```

6. Verify that all of the Nova services on all Nodes are operational:

```
nova-manage service list
Binary           Host          Zone   Status     State Updated_At
nova-volume      <your_host>   nova   enabled    :-)   2012-06-06 22:30:05
nova-consoleauth <your_host>   nova   enabled    :-)   2012-06-06 22:30:04
nova-scheduler   <your_host>   nova   enabled    :-)   2012-06-06 22:30:05
nova-compute     <your_host>   nova   enabled    :-)   2012-06-06 22:30:02
nova-network     <your_host>   nova   enabled    :-)   2012-06-06 22:30:07
nova-cert        <your_host>   nova   enabled    :-)   2012-06-06 22:30:04
```

7. Run the nova_test script from the Controller that has the openstack::test_file class (Control1 by default):

```
. /tmp/test_nova.sh
```

8. Connect to the Horizon Virtual IP address (OpenStack Dashboard) through a web browser:

- create a keypair
- edit the default security group to allow icmp -1 -1 and tcp 22 22 for testing purposes.
- fire up a VM.
- create a volume.
- attach the volume to the VM.
- allocate a floating IP address to a VM instance.
- verify that volume is actually attached to the VM and that it is reachable by its floating ip address.
- The above steps are well documented in the Essex Compute Admin Guide

Administration

The OpenStack Cisco Edition includes several tools to assist with administration. The clean_node.sh script is a tool that removes the necessary configurations associated to a node and starts the rebuilding process. The script is located at <module_path>/os-docs/examples/clean_node.sh

- Usage Example:

```
/etc/puppet/modules/os-docs/examples/clean_node.sh control01.corp.com
```

- **Note:** The script will use sdu.lab if you do not specify the FQDN. You can change the default domain name by editing the clean_node.sh script.

Additional administration guides are available for providing details on managing and operating an OpenStack environment.

Participating

Need a feature? Found a bug? Let us know!

We are extremely interested in growing a community of OpenStack experts and users around these modules so they can serve as an example of consolidated best practices of production-quality OpenStack deployments.

The best way to get help with this set of modules is through email:

openstack-support@cisco.com

Issues should be reported here:

openstack-support@cisco.com

The process for contributing code is as follows:

- fork the projects in github
- submit pull requests to the projects containing code contributions

Future features:

Efforts are underway to implement the following additional features:

- Support OpenStack Folsom release
- These modules are currently intended to be classified and data-fied in a site.pp. Starting in version 3.0, it is possible to populate class parameters explicitly using puppet data bindings (which use hiera as the back-end). The decision not to use hiera was primarily based on the fact that it requires explicit function calls in 2.7.x
- Integrate with PuppetDB to allow service auto-discovery to simplify the configuration of service association