Instructions from:

https://github.com/CiscoSystems/cisco-openstack-docs repository

Contents

- 1 Simple Openstack
  - 1.1 Infrastructure Model
  - 1.2 Building the environment
  - 1.3 SKIP THIS SECTION IF YOU RAN THE CURL COMMAND
  - 1.4 CONTINUE HERE IF YOU RAN THE CURL COMMAND
  - 1.5 Testing OpenStack
  - 1.6 Adding in Swift

Simple Openstack

Infrastructure Model

The test environment is build on Cisco UCS B-200 M2 blades in a Cisco Blade chassis with dual UCS 6100 switches and Cisco VIC adapters. The example site.pp uses the Flat_DHCP model, and assumes that an eth0 interface will be attached to the "public" newtork, and that an eth1 interface will be avaialble for the Flat network bridge.

The blade config is:

2x 6 core processors 96GB memory Cisco VIC 2x 300GB SATA drives in a Raid-1 configuration

The system profile has two VNICs, one on VLAN 100 (arbitrary), and one on VLAN 105 (also arbitrary). VLAN 100 is internet accessible via a NAT gateway and via inbound VPN via an ASA firewall. VLAN 105 is unconnected

"Public" addresses on VLAN100 are in the 192.168.100.0/24 range, and a static pool (nova pool) is carved out of that range as 192.168.100.64/28 "Private" addresses are from the floating pool in the 10.0.0.0/16 range, but a network in the 10.0.0.0/24 range is actaully assigned to the openstack tenant.

Building the environment

1. Build an Ubuntu 12.04 system.

We still need a build node (though will plan to migrate to a RAMFS based system soon), and you can use the preseed file in the example directory to build the base os. Start with the ISO boot (or USB boot), and at the initial installer screen, after you pick your language, hit F6 (or FN-F6 on a mac), and ESC, and add (don't delete anything from the current boot string):

priority=critical locale=en_US url=http://128.107.252.163/preseed

If you can't get to the 128 address (i.e., you're outside of Cisco), host the preseed file on your own machine. You may need to add network information (unless you have DHCP enabled, which you may want to disable and give control over to cobbler).

If you don't want to go the pre-seed route, then build a machine normally, make sure you have ~10GB of disk space available, and I recommend using an LVM volume rather than just using the whole disk, but it's less critical for this build.

Once the node is built log in (localadmin:ubuntu are the default), and become root (I usually do "sudo -H bash"), or preface all the following commands with "sudo".

```
apt-get install curl
bash < <(curl -s -k -B https://raw.github.com/CiscoSystems/cisco-openstack-docs/master/examples/bu
```

This should get you to a state where puppet is ready to be configured (by editing the config scripts).

SKIP THIS SECTION IF YOU RAN THE CURL COMMAND

Alternatively, you can follow the same basic steps as the script above:

```
apt-get update && apt-get dist-upgrade -y
```

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

Next, add the Cisco Mirror repository:

```
apt-get install -y python-software-properties
add-apt-repository -y ppa:cisco-openstack-mirror/cisco
apt-get update
```

Now that the Cisco Mirror repo has been added, install the Cisco Puppet Modules (installed into /usr/share/puppet/modules):

```
apt-get install puppet-openstack-cisco
```

Optionally, you can install the following packages.

Documentation:

```
apt-get install cisco-openstack-docs
```

Utilities:

```
apt-get install cisco-openstack-utils
```

You will need a couple additional packages:

```
apt-get install ntp puppet git ipmitool -y
```

If your cisco-openstack-docs package does not include a sample site.pp and cobbler-node.pp, then pull cisco-openstack-docs directly from the Github repository:

```
git clone https://github.com/CiscoSystems/cisco-openstack-docs ~/os-docs
```

Then you need to set up your site:

```
cp ~/os-docs/examples/{site.pp,cobbler-node.pp} /etc/puppet/manifests/
```

CONTINUE HERE IF YOU RAN THE CURL COMMAND

YOU MUST THEN EDIT THESE FILES. They are fairly well documented, but please comment with questions. You can also read through these descriptions: Cobbler Node and Site

Then 'puppet apply' it:

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

I recommend a reboot at this point, as it seems that the puppetmaster doesn't restart correctly otherwise.

And now you should be able to load up your cobbled nodes:

```
~/os-docs/examples/clean_node.sh {node_name} example.com
```

or if you want to do it for *all* of the nodes defined in your cobbler-node.pp file:

```
for n in `cobbler system list`; do ~/os-docs/examples/clean_node.sh $n example.com ; done
```

note: replace example.com with your nodes proper domain name.

Testing OpenStack

Once the nodes are built, and once puppet runs (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: user: localadmin, password: ubuntu. If you SU to root, there is an openrc auth file in root's home directory, and you can launch a test file in /tmp/nova_test.sh.

You should now have a cirros image and a running instance (called dans_vm if you didn't change anything).

Adding in Swift

Swift adds a scaleable redundant object storage system to the openstack environment, and is a core part of the general model. In order to add swift, you will need a minimum of 3 additional machines, prefereably with a large number of fast hard drives, with RAID preferably disabled (that's right RAID is not recommended for Swift clusters).

Since the system does not yet catalog devices, you will need to manually define the devices to be used by the swift system and configure the appropriate "nodes" and "rings" for the number of devices defined. The included example is set up for two devices, either LVM based devices (useful if you are using UCS blades which nominally only support a maximum of 2 drives today or are working in a constrained development environment) or perferably raw direct access devices. The puppet modules will attempt to format the entire device as an XFS file system, and then add them in to the system.

EDIT THE swift-nodes.pp FILE before including it in your site.pp file. Then you can add the swift-nodes.pp file into your site.pp (or uncomment it from the example file):

```
import "swift-nodes"
```

Build/re-build the swift machines, and you should have a swift capable system.