Contents

Deployment Overview

THIS DOCUMENT IS A WORK IN PROGRESS.

The All-in-One method of deploying OpenStack will give the user a functional OpenStack environment with the current general deployment components based on the Cisco Edition model. Currently, this deployment model includes:

- Nova
- Glance
- Keystone
- Quantum
- Nova-Volume
- Horizon

It does not include:

- Cinder
- Swift
- Other incubation Projects

It does include a subset of non incubated but still useful tools:

- Nagios
- Collectd/Monitd

Deployment Prerequisites

You should already have a single machine on which to build your openstack environment, and it should have access to the public internet (though it is not a requirement for the public internet to have access to the machine). This machine can be a physical system, or a virtual instance. You should have ~30GB of Free disk space, and you will likley want at least 4GB of RAM, though more is almost always better. It is also a requirement currently that you run Ubuntu 12.04.01 though 32 or 64 bit should not make a difference (we've tested exclusively on 64 bit OSs). It may also be possible to leverage Ubuntu Server 12.10 or a RedHat variant (RHEL/RHEV 6.3, Fedora 17, Centos 6.3), but these have not been tested. As this system will use a single interface, the upstream network is not particularly critical, though it is certainly easier to work with the system if there is a consistent CIRD subnet IPv4 address block availalble. IPv6 _should_ also function, but has not been tested in the current system.

- x86 class server or virtual machine with >=4GB RAM, >=30GB disk

- Ubunt Server 12.04.1 installed with OpenSsh-server installed
- A single network interface, preferably with an open CIDR subnet attached (e.g. 192.168.100.1/26)

Deployment Steps

1. Install Ubuntu 12.04.1 x86_64 Server operating system

    - Follow the standard install process, selecting the defaults
    - Install at least OpenSSH-server
    - We have tested principally with the disk in LVM mode

2. Install additional prerequisite packages:

    - puppet, git

Become root

```
$ sudo -H bash
```

Install packages

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

Now, clone the puppet baseline code that will help bring in the rest of the system.

```
# git clone https://github.com/robertstarmer/folsom-manifests -b all-in-one all-in-one-manifests
# cp all-in-one-manifests/manifests/* /etc/puppet/manifests/
```

Note that if you need to go through a proxy server to get to the internet, you may need to pass appropriate environment variables to these commands like so:

```
# http_proxy=http://myproxy:port https_proxy=https://myproxy:port apt-get update && apt-get dist-u
# https_proxy=https://myproxy:port git clone https://github.com/robertstarmer/folsom-manifests -b
```

Now grab the rest of the puppet code:

```
# cd /etc/puppet/manifests; ./puppet-modules.sh
# # Or, if you need to use a proxy:
# # cd /etc/puppet/manifests; https_proxy=https://myproxy:port ./puppet-modules.sh
```

Next we need to resolve a few "glitches" in the current code:

```
# for n in nova quantum openstack horizon;
# do
# rm -rf /etc/puppet/modules/$n
# git clone -b folsom https://github.com/robertstarmer/puppet-$n /etc/puppet/modules/$n
# done
```

Here again, if you need to go through a proxy server to get to the internet, you may need to modify the command like so:

```
# for n in nova quantum openstack horizon;
# do
# rm -rf /etc/puppet/modules/$n
# https_proxy=https://myproxy:port git clone -b folsom https://github.com/robertstarmer/puppet-$n
# done
```

Now. You will need to edit the sample site.pp to meet your specific site. This should require changing only a very small number of parameters:

- node hostname (recommend something with "aio" in the name)
- Default interface IP, Netmask, and default gateway
- upstream ntp server
- do you need a proxy for internet access?

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

System Fixes

With those steps completed, there are only a few small tweaks that still need to be made to the system before we can launch the puppet. 1. Because we are going to set up Open Virtual Switch as a part of the deployment, and because we are going to only leverage a single interface, we will need to modify the network interfaces file. this may be resolved in a future deployment model, but today, this is a part of the basic setup:

```
# vi /etc/network/interfaces
```

the simplest way to fix this file is to copy the two lines for your primary interface (often this is eth0), and paste them back in.

An example before:

</pre>auto eth0 iface eth0 inet static

```
  address 1.2.3.4
  netmask 1.0.0.0
  dns-...
  etc.
```

</pre>

An example after:

```
auto eth0
iface eth0 inet static
   address 0.0.0.0

auto br-ex
iface br-ex inet static
   address 1.2.3.4
   netmask 1.0.0.0
   dns-...
   etc.
```

And because OpenVirtual Switch starts later than the "rest" of the network subsystems, we'll fix a glitch that would add a few minutes to the boot time of the server. Find the section in the file: /etc/init/failsafe.conf that looks like:

```
        $PLYMOUTH message --text="Waiting for network configuration..." || :
         sleep 40

         $PLYMOUTH message --text="Waiting up to 60 more seconds for network configuration..." || :
```

```
        sleep 59
        $PLYMOUTH message --text="Booting system without full network configuration..." || :
```

and make it look like:

```
        $PLYMOUTH message --text="Waiting for network configuration..." || :
        $PLYMOUTH message --text="Waiting up to 60 more seconds for network configuration..." || :
        $PLYMOUTH message --text="Booting system without full network configuration..." || :
```

Deploy!

Great, now we're ready for the majic to happen!

Important Note: unlike in the previous steps, if you're behind a proxy server you don't want to pass around proxy environment variables here. If you've set up proxy support in your site.pp file, things that need to talk to the internet (e.g. package installation processes) will use the proxy settings you added to site.pp. If you do pass in proxy environment variables, you may have some problems due to the fact that the puppet code you're about to invoke makes REST API calls--and if those are mistakenly sent through a proxy server, they may not reach their intended destinations.

First, we can do a quick test to make sure that our site.pp doesn't contain any obvious errors:

```
# cd /etc/puppet/manifests
# puppet apply -v --noop site.pp
```

There may still be some errors thrown because things aren't actually installing, but it shouldn't quit while trying to compile the catalog.

Now, let's run for real. I tend to run this step inside of screen, to make sure it completes (note: you may need to run "apt-get install screen" if screen isn't installed already), even though I'm going to ask the system to reboot after completion so that our new network configuration takes place (assuming a single network interface).

```
# screen
# cd /etc/puppet/manifests
# puppet apply -v site.pp ; reboot
```

In about 15 minutes or so (depending on the speed of your disk, network connection, etc.) you should have a basic OpenStack system based on the Cisco Edition model!.

Now, how do you use that brand new system? Well, first you will need to load some kind of image into the environment, and we can suggest the Cirros image as it's small and yet functional enough to get started. Certainly you can deploy other cloud images (including Amazon images!) into the system, but be aware that your images, and your deployed VMs will be sharing whatever local disk space your system has. One reason we like the Cirros image is that it's only ~10MB in size!. You can grab a copy from here: http://download.cirros-cloud.net/0.3.1/

A quick way to get a first image up is to use our Quantum test script:

```
# git clone https://github.com/CiscoSystems/quantum-l3-test
# # Or if you're behind a proxy:
# # https_proxy=https://myproxy:port git clone https://github.com/CiscoSystems/quantum-l3-test
# cd quantum-l3-test
```

You can use the IP range of our external network (the one we used to tweak the network configuration earlier) though you will need to have 3 free addresses usually right above the router address (so if the router is 192.168.100.1, you would want to make sure that .2 .3 and .4 are available). It is also good to know what your local DNS server is so that DNS can be assigned to the new vm when it starts. And lastly, you'll want a pointer to the Cirros image (or your image of choice). The system expects a qcow2 complete image (sorry, it doesn't load the 3 parts of an AMI at this point). The direct path to the Cirros image: http://download.cirros-cloud.net/0.3.1/cirros-0.3.1-i386-disk.img

Now you can just run:

```
# cd quantum-l3-test
# ./create_vm
# # Or, if you're behind a proxy:
# # https_proxy=https://myproxy:port http_proxy=http://myproxy:port no_proxy="mydomain.com" ./crea
```

If all goes well, you should be able to do the following (assuming a 192.168.1.0/24 is your "external" network):

```
# ssh cirros@192.168.1.4
```

Conclusions

There's certainly a lot more that you can do with this system now that it's running, but hopefully this process has removed a lot of the overhead of getting a basic system online.