**Openstack Deployment Guide for OCI**

**We followed, abridged, and skipped around the guide laid out at:** <http://docs.openstack.org/developer/openstack-ansible/install-guide/index.html>

**First, we need to identify the allocation of the rack and servers we desire to use. We would like to keep the ip’s chosen for our elements in a contiguous range. For our case, we designed our setup as follows:**

10.241.1.54 → Storage/Cinder

10.241.1.55 → Infrastructure 1

10.241.1.56 → Infrastructure 2

10.241.1.57 → Infrastructure 3

10.241.1.58 → Compute Node 1

10.241.1.59 → Logging/HA Proxy

**This arrangement is the bare minimum needed for an iteration of an openstack cloud. Three infrastructure nodes are required for Galera to operate in a highly available manner. Otherwise, a single infrastructure node can be used. Generally speaking, this means Galera will operate best with an odd number of infrastructure nodes. Then, we need to verify if the 10-Gigabit network interface is enabled for each node. This is needed for our servers to talk to each other. We also need to check if they have a 1 gigabit network interface as well for external communication. Use the following command to determine the hardware configuration for each node:**

$ lshw -C Network

**Determine which interfaces match the MAC address of 10-Gigabit network interface using the following:**

$ ip a

**The output for the previous command will give a lot more information than is needed for our purposes, but the pertinent information within will be as the following:**

10.241.1.54 Storage eth1 is 10G

10.241.1.56 In the Infra 2 case, the **eth2** is 10-Gigabit network interface.

10.241.1.58 Compute eth1 is 10G

10.241.1.59 Logging/HAProxy eth1 is 10G

**For each node, we need to ensure that the 10-Gigabit interface is up, if it is down (note that we use eth2 for our case. replace with the interface that corresponds to your 10g interface):**

$ ip link set dev eth2 up

**We then use the following command to verify the current state of the interface (up or down). It is a best practice to always verify that your work is completed even after having just recently executed a command. This is done for each node as well:**

$ ip a s <interface>

**Create key pairs. Key pairs are like the guest keys to your friend/family’s house. For our case, we need to take the “public key” from all our infra nodes and copy them to all other nodes. Which would mean generating a public key for infra1, infra2, and infra3, copying those public keys to infra1, infra2, infra3, storage, compute, and the logging node.**

$ ssh-keygen -t rsa -f cloud-key

**br-vxlan**

$ ssh root@10.241.1.54 "echo '`cat ~/.ssh/cloud-key.pub`' >> ~/.ssh/authorized\_keys"

**We then installed “dsh” (Distributed Shell) to allow us to send commands to several nodes at once by using the following command:**

$ apt-get install dsh -y

**After installing, we used “dsh” (Distributed Shell) to test the connection from Infras to other nodes using this procedure:**

1. **Create a directory called group:**

$ mkdir -p /root/.dsh/group

**2.   Create a file named hosts and use the editor of your choice to input the ip’s of the hosts:**

$ vi /root/.dsh/group/hosts

**Following the above steps, we send the command “echo hi” to all the ip’s contained within the file hosts while showing their respective ip’s in the CLI:**

$ dsh -g hosts -c -M “echo hi”

**After ensuring the communication to all hosts, we need to determine the kernel version of each host using:**

$ dsh -c -g hosts -M ‘uname -a’  # (It gives the kernel version)

(Side note-- if your kernel version is version 34 or less, it needs upgrading. otherwise, leave it alone)

**We can then update all hosts if needed:**

$ dsh -c -g hosts -M “apt-get update”

***Access the installation guide from Rackspace:*** docs.openstack.org/developer/openstack-ansible/

**After the update, we can then install several packages needed for further work. If this or any other later steps fail, it may be because of the \ in the script. Simply delete the \ to see if that works:**

$ apt-get install aptitude build-essential git ntp ntpdate \  
 openssh-server python-dev sudo

**Next, we are going to install ansible so we can use it to install playbooks later on. Navigate to opt directory:**

$ cd /opt

**Then, acquire the openstack ansible program found at their git repo:**

$ git clone <https://github.com/openstack/openstack-ansible.git>

**Access the openstack-ansible folder:**

$ cd openstack-ansible/

**Execute the bootstrap-ansible.sh script to install ansible:**

$ ./scripts/bootstrap-ansible.sh

**Verify Ansible version by:**

$ ansible --version (optional)

**The following command installs several packages needed for the next steps (LVM and networking related) to all the hosts detailed in our previously defined hosts file:**

$ dsh -c -g hosts -M “apt-get install bridge-utils debootstrap ifenslave ifenslave-2.6 lsof lvm2 ntp ntpdate openssh-server sudo tcpdump vlan -y”

End of the Day 2…...STOP

**LVM Settings**

**In order to set up our storage node and containers, we need to determine the current condition of the node. We ssh’ed into our storage node as follows:**

$ ssh root@10.241.1.54

**Then, we check the current partitions available using:**

$ fdisk -l

**The above is a lowercase L. It will list the partitions for all devices unless you provide a 3rd argument, which would point fdisk to the specific device you desire to list the partitions of.**

**Then, we use the following to display the current existing physical volumes:**

$ pvdisplay

**After you have found that there are no current physical volumes and have identified the hard drive partitions available, we use the following command to create the physical volume:**

$ pvcreate /dev/sdb1

**We then use the following command to view our current partitions:**

$ fdisk /dev/sdb

**Then, we need to create our volume group, naming it cinder-volumes:**

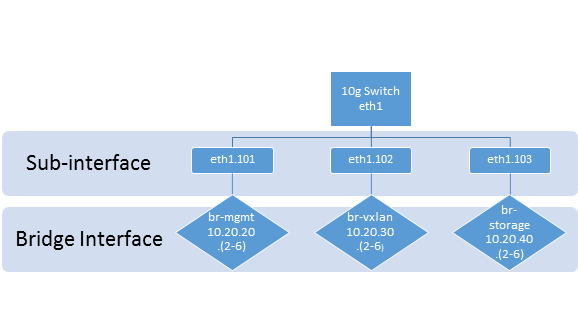
$ vgcreate cinder-volumes /dev/sdb1

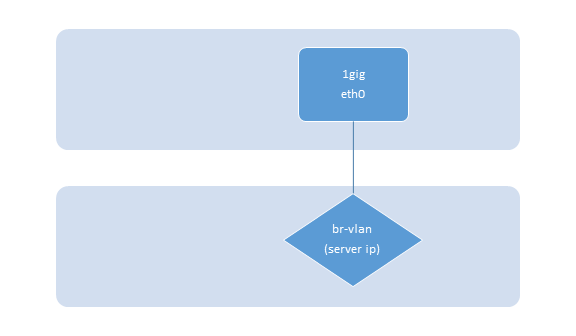
**Finally, we display the volume groups to ensure our work was done correctly:**

$ vgdisplay

**Configuring the Network**

**VLAN Topology**

****

****

**The two topologies above are a brief overview of what our network will look like. Each node (Storage, HAProxy, Infras, Compute) will need to have certain connections that will allow the communication between the various nodes. The bridges are labelled as br- and followed by the type of connection. Each bridge connects to the other nodes via their respective bridge connections. The second topology describes the difference between the VLAN bridge and the other 3. In this case our VLAN bridge has no subinterface between it and the external network. This is because the external physical network is not set up to direct that type of traffic.**

**The network interface settings that we used reside here:**

<https://gist.github.com/Frank-ZhangXin/76a98d7f37c292f98827>

**\*NEED to add to OCI github later**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Added Start - Gonzalo \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**In order to define our setting for VLAN we require the following:**

1. **Netmask**
2. **Gateway**
3. **Dns-nameservers**

**In order  to obtain the netmask, use:**

$ route

Kernel IP routing table

Destination     Gateway         Genmask         Flags Metric Ref    Use Iface

default         10.241.254.253  0.0.0.0         UG    0      0        0 eth0

10.241.0.0      \*               **255.255.0.0**     U     0      0        0 eth0

**In order to obtain the gateway, use:**

$ ip r

default via **10.241.254.253** dev eth0

10.241.0.0/16 dev eth0  proto kernel  scope link  src 10.241.1.55

**In order to obtain the dns-nameserver, look into the following file by:**

$ vi /etc/resolv.conf

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Added End - Gonzalo \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Interface settings:**

**Navigate to “/etc/network/interfaces” and edit it to resemble the file that follows this instruction:**

$ cd /etc/network/interfaces

**For reference, in the interface file eth0 is 1G and eth1 is 10G:**

# The loopback network interface

auto lo

iface lo inet loopback

# The primary network interface

auto eth0

iface eth0 inet manual

# auto automatically brings the interface up at boot

auto eth1

iface eth1 inet manual

# the manual variable sets it so that the ip settings are not directly configured here, but rather will # be configured by external scripts or files

iface eth1.101 inet manual

 vlan-raw-device eth1

iface eth1.102 inet manual

 vlan-raw-device eth1

iface eth1.103 inet manual

 vlan-raw-device eth1

auto br-mgmt

iface br-mgmt inet static

 bridge\_stp off                **# spanning tree protocol enables multiple bridges to work together**

 bridge\_waitport 0           **# maximum wait time for specified ports to become available**

 bridge\_fd 0                     **# forwarding delay**

 bridge\_ports eth1.101

 address 10.20.20.2

 netmask 255.255.255.0

auto br-vxlan

iface br-vxlan inet manual

 bridge\_stp off

 bridge\_waitport 0

 bridge\_fd 0

 bridge\_ports eth1.102

 # On infra nodes, IP address is in neutron container

auto br-vlan

# the bridges for vlan, storage, and management are set to static in order to make their definitions # unchangeable by scripts and or

iface br-vlan inet static

 address 10.241.1.55

 netmask 255.255.0.0

 gateway 10.241.254.253

 dns-nameservers 10.241.1.1

 # Neutron tags this traffic with l3 headers

 bridge\_stp off

 bridge\_waitport 0

 bridge\_fd 0

 bridge\_ports eth0

auto br-storage

iface br-storage inet static

 bridge\_stp off

 bridge\_waitport 0

 bridge\_fd 0

 bridge\_ports eth1.103

 address 10.20.40.2

 netmask 255.255.255.0

**The next set of steps involve bringing the recently configured network online and then checking the status. This is the part that will contain some debugging/error correction. <We make a lot of assumptions here as to the experience of the reader.>**

**Bring up interface eth1(10G):**

$ ifup eth1

**Check eth1 status with:**

$ ip a s eth1

**Bring up sub-interface (eth.101, 102, and 103):**

$ ifup eth1.101

**You should get 8021q module needed warning and system will add it to kernel:**

**The output for the command**$ ip a s eth1.101 **should appear like the following:For eth1.101:**

WARNING:  Could not open /proc/net/vlan/config.  Maybe you need to load the 8021q module, or maybe you are not using PROCFS??

Set name-type for VLAN subsystem. Should be visible in /proc/net/vlan/config

Added VLAN with VID == 101 to IF -:eth1:-

**The above warning is only because this is the first time that the interface has been added. Nothing to be concerned about.**

**The output below is similar to the above except that it is missing the warning. Some interfaces may give a warning and others not, but so long as they indicate that they are added to VID corresponding to their respective ethernet designation, eth1.102 in this case, then everything is fine. For eth1.102:**

Set name-type for VLAN subsystem. Should be visible in /proc/net/vlan/config

Added VLAN with VID == 102 to IF -:eth1:-

**Same story as above. This is just another example to display how the VID corresponds to our interface. For eth1.103:**

Set name-type for VLAN subsystem. Should be visible in /proc/net/vlan/config

Added VLAN with VID == 103 to IF -:eth1:-

**Test all servers with the following command and verify that their state is up using the following command. The output should appear similar to what follows the command:**

$ ip a

..

5: eth1.101@eth1: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc noqueue state **UP** group default

   link/ether 90:e2:ba:00:26:14 brd ff:ff:ff:ff:ff:ff

   inet6 fe80::92e2:baff:fe00:2614/64 scope link

      valid\_lft forever preferred\_lft forever

6: eth1.102@eth1: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc noqueue state **UP** group default

   link/ether 90:e2:ba:00:26:14 brd ff:ff:ff:ff:ff:ff

   inet6 fe80::92e2:baff:fe00:2614/64 scope link

      valid\_lft forever preferred\_lft forever

7: eth1.103@eth1: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc noqueue state **UP** group default

   link/ether 90:e2:ba:00:26:14 brd ff:ff:ff:ff:ff:ff

   inet6 fe80::92e2:baff:fe00:2614/64 scope link

      valid\_lft forever preferred\_lft forever

**‘8021q’ module is necessary for sub-interface setup, check if it exists in your kernel:**

$ lsmod | grep 802

**The output should resemble the following:**

**8021q**                  28933  0

garp                   14384  1 8021q

mrp                    18778  1 8021q

**Interface settings - bridges:**

**We need to create a series of bridges that allow communication to flow without conflict. For instance, we create a br-storage bridge so that storage traffic, meaning requests or commands pertaining to cinder, to and from the storage node can be done along that path. The same goes for each type of service on each node. The storage node’s communications to the infrastructure node may go through the management bridge if the nature of the communication is networking service related. Once we have created each bridge that we need, we have to associate the bridges with certain VLANs so that they know what VLANs traffic is allowed on each respective bridge.**

**To Show all bridges on the network:**

$ brctl show

**To Activate the Bridge:**

$ ifup br-mgmt

**To check the status of a specific network construct:**

$ ip a s br-mgmt

**To check the configuration of all the bridges:**

$ dsh -g hosts -M ‘brctl show’  (Need to be ssh’ed into an Infra Node)

**To ping from Infra node to various bridges:**

ping -I br-storage 10.20.40.3

**The above command pings from the first variable (br-storage) to 10.20.40.3 in order to verify communication. It does this ping along the br-storage bridge’s connections (veths). The source address gets changed to that of the address on br-storage.**

**Configure VLAN on Cisco Nexus Switch (Just for OCI)**

**We ssh’ed into the cisco switch, check the current VLANs that are created, create our own VLAN using cisco command :** vlan 101**. After creating the VLAN we name it using:** “name *br-mgmt”***, (removing the quotes) which in this case names it br-mgmt.**

**We then checked which ports are currently associated with the VLAN, which there may not be any, and associated the necessary ports.The series of commands that we used to perform these steps are as defined below our tables below.**

**We created the table below that corresponds to our switches ports and the interface designation as well.**

|  |  |  |
| --- | --- | --- |
| IP | 10G | Nexus Switch Port |
| 10.241.1.54 | eht1 | Eth1/26 |
| 10.241.1.55 | eht1 | Eth1/27 |
| 10.241.1.56 | eht2 | Eth1/21 |
| 10.241.1.58 | eht1 | Eth1/19 |
| 10.241.1.59 | eth1 | Eth1/17 |

**This table details our network for all nodes and bridges.**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Storage/Cinder | Infra 1 | Infra 2 | Compute | Logging/HA Proxy |
| Server | 10.241.1.54 | 10.241.1.55 | 10.241.1.56 | 10.241.1.58 | 10.241.1.59 |
| Br-mgmt | 10.20.20.6 | 10.20.20.2 | 10.20.20.3 | 10.20.20.4 | 10.20.20.5 |
| Br-vlan | 10.241.1.54 | 10.241.1.55 | 10.241.1.56 | 10.241.1.58 | 10.241.1.59 |
| Br-vxlan | 10.20.30.6 | 10.20.30.2 | 10.20.30.3 | 10.20.30.4 | 10.20.30.5 |
| Br-storage | 10.20.40.6 | 10.20.40.2 | 10.20.40.3 | 10.20.40.4 | 10.20.40.5 |

**When privileges are elevated, we are allowed to go to a global configuration mode using:**

$ config t

**Then, we use the command** $ show vlan **to show the current VLANs that are arranged on the switch.The exact sequence and arrangement of steps that follow this part depend entirely on the current configuration of the switch you are using.**

**To Delete an existing  VLAN:**

$ no vlan <vlan-number>

**To create a new VLAN:**

$ vlan <vlan-number>

**To name the VLAN:**

$ name <vlan name>

**To start the VLAN:**

$ state active

$ no shutdown

**To exit from this VLAN:**

$ exit

**To see interface status:**

$ show interface brief

**To configure a port to a VLAN we have to enter into that port number using the following:**

$ interface ethernet <port number> (1/20 for example)

**To eliminate the port from VLAN:**

$ interface ethernet <port number> (1/20 for example)

$ no switchport mode trunk

**We set the port to trunk mode in order to have traffic from outside the assigned port.To set port mode (if a port is not under trunk mode we need to assign it this):**

$ switchport trunk mode

**To see specifications of a particular port number:**

$ show running-config interface Ethernet <port number> (1/20 for example)

**To remove a specific port from an interface:**

$ no switchport trunk allowed vlan <port number>

**To add a specific port to the  interface:**

$ switchport trunk allowed vlan <port number>

**The following is an example of the series of commands that we would run to create a VLAN for our purposes:**

**Creation of VLANs:**

$ vlan 201          # Creates a VLAN with ID 201

$ name br-mgmt2     # We renamed it br-mgmt2

$ state active      # This tells the switch to make the VLAN active

$ no shutdown       # This command tells the switch to turn on the VLAN

$ exit              # Exits this interface

$ vlan 202          # Repeats the same process from above

$ name br-vxlan2

$ state active

$ no shutdown

$ exit

$ vlan 203

$ name br-storage2

$ state active

$ no shutdown

$ exit

**Check interfaces’ status:**

$ show interface brief

**When we first checked the configurations in our cisco switch we found that there were already some VLANs assigned to the naming convention that we used. We then had to remove some of the ports from our VLANs and add others. The process went similar to as follows.**

**Remove interfaces 17, 19 from all VLANs associated with these ports**

**Interface 17**

$ interface ethernet 1/17

$ no switchport mode trunk

**Interface 19**

$ interface ethernet 1/19

$ no switchport mode trunk

**Associate our new VLANs with all of our new physical switch ports (interfaces) connected to our nodes.**

**Interface 17**

$ interface ethernet 1/17

# Remove unnecessary association of this interface from other VLANs.

# $ no switchport trunk allowed vlan <VLAN #>

$ show running-config interface Ethernet 1/17

# If Interface 17 is not in trunk mode

# $ switchport mode trunk

$ switchport trunk allowed vlan 201-203

$ exit

**Interface 18**

$ interface ethernet 1/18

$ show running-config interface Ethernet 1/18

$ switchport trunk allowed vlan 201-203

$ exit

**Interface 19**

$ interface ethernet 1/19

$ show running-config interface Ethernet 1/19

$ switchport trunk allowed vlan 201-203

$ exit

**Interface 21**

$ interface ethernet 1/21

$ show running-config interface Ethernet 1/21

$ switchport mode trunk

$ switchport trunk allowed vlan 201-203

$ exit

**Interface 26**

$ interface ethernet 1/26

$ show running-config interface Ethernet 1/26

$ switchport mode trunk

$ switchport trunk allowed vlan 201-203

$ exit

**Interface 27**

$ interface ethernet 1/27

$ show running-config interface Ethernet 1/27

$ switchport mode trunk

$ switchport trunk allowed vlan 201-203

$ exit

**To make the changes permanent:**

$ copy running-config starting-config

**To remove an association from VLAN interface**

$ brctl delif br-vxlan eth1.102

**To remove a VLAN interface**

$ vconfig rem eth1.102

**To bring down a bridge**

$ ip link set br-vxlan down

**To delete a bridge**

$ br-vxlanbr-vxlan

**Here is an example of the sequence of commands that we had to use:**

$ brctl delif br-mgmt eth1.101

$ vconfig rem eth1.101

$ ip link set dev br-mgmt down

$ brctl delbr br-mgmt

$ brctl delif br-vxlan eth1.102

$ vconfig rem eth1.102

$ ip link set dev br-vxlan down

$ brctl delbr br-vxlan

$ brctl delif br-storage eth1.103

$ vconfig rem eth1.103

$ ip link set dev br-storage down

$ brctl delbr br-storage

**Configuring Ansible**

**To configure our containers to be able to talk to outside world as well as each other, we edited the file** openstack\_user\_config.yml **found in the directory:** /etc/open-stack\_deploy/ **. The beginning of the file has a section titled cidr\_networks.**

**For each of those subsections (container, tunnel, and storage) we have to assign a non private ip range. We chose to use** 10.20.20.0/24 **for container network (which is the same as the management network),** 10.20.30.0/24 **for the tunnel network (which is the vxlan network), and** 10.20.40.0/24 **which is the storage network.**

**The next section is title used\_ips. This section tells ansible what ip ranges are currently in use and not to use them. We replaced that section with the following:**

* “10.20.20.2,10.20.20.50”
* “10.20.30.2,10.20.30.50”
* “10.20.40.2,10.20.40.50”

**The above gives a range of ips from 2-50 so that there is room for expansion. This set range does not need to be this large necessarily.**

**The next section is titled** global\_overrides **which we will replace the subsections** internal\_lb\_vip\_address **with our internal address of 10.20.20.10 and then replace** external\_lb\_vip\_address **with 10.241.1.59. The internal ip address was arbitrarily chosen from the range for our mgmt network, while the external address is the address of our HAProxy/Logging node. The two subsections that follow these are only to be changed if they are different than what is configured in our hosts (tunneling\_bridge, management\_bridge).**

**The next section that we need to modify is the section for how to configure a VLAN for physical provider network. The first subsection you find that describes VLAN has** type: “flat” **and** net\_name: “flat”**. Leave that section alone and the following section details what was described just previous, setup for a VLAN for physical provider network. We took the entire section beginning with the subheading** -network **and deleted everything including the subsection** -linux\_bridge\_agent**.**

**The next section that we needed to modify is titled shared infra hosts. We need to modify the following section to appear as follows:**

shared-infra\_hosts:

**<begin delete>**

aiol:

  #rabbitmq, and galera are set to multiples to test clustering

  affinity:

     galera\_container: 3

     rabbit\_mq\_container: 3

  ip: 172.29.236.100

**<end delete>**

**For our case, we replaced the above deleted section with the following:**

infra1:

  ip: 10.20.20.55

infra2:

  ip: 10.20.20.56

infra3:

  ip: 10.20.20.67

os-infra\_hosts:

**<begin delete>**

aiol:

  # Horizon is set to multiple to test clustering. This test only requires x2.

  affinity:

     horizon\_containter: 2

  ip: 172.29.236.100

**<end delete>**

**For our case, we replaced the above deleted section with the following:**

infra1:

  ip: 10.20.20.55

infra2:

  ip: 10.20.20.56

infra3:

  ip: 10.20.20.67

**Repeat the same operation for** storage-infra\_hosts, repo-infra\_hosts, identity\_hosts, and network\_hosts**. Delete the body of the section and replace it with the infra1, infra2, and infra3 ips as was demonstrated above.**

**Next, the compute hosts are different. Compute host ips are the same as your previously assigned compute hosts. For our case, we deleted the body as above, but replaced the body with our single compute host ip:** 10.241.1.58**.**

**The last sections we are going to edit are** log\_hosts and haproxy\_hosts**. We delete their respective bodies as before and since our HAProxy node is on the same server as our Logging node, we edit them both to contain the following:**

haproxy\_log:

     ip: 10.241.1.59

**This concludes the editing of this file. Save and quit. Ansible now has been informed as to where everything is to reside. Next, we are going to use a python script to auto-fill the file** user\_secrets.yml **with the passwords required for the various service credentials.**

**Autofilling user\_secrets**

**First, we navigate to the directory** /opt/openstack-ansible/scripts **and run the python script that will generate the passwords needed by the following command:**

$ python pw\_token\_gen.py --file  /etc/openstack\_deploy/user\_secrets.yml

**Then, verify that** user\_secrets.yml **has been modified by opening it with your editor of choice. We chose to also go ahead and modify the line** keystone\_auth\_admin\_password **to contain the password** openstack**. This was to avoid having to type in that long string that is automatically generated by the python script every time it was needed later. We can always choose much harder passwords if needed.**

**Final ansible configuration and syntax check**

**Next, we checked the file** user\_variables **located in the directory** /etc/openstack\_deploy/ **to make sure that the line for** nova\_virt\_type: kvm **is commented out as follows:**

# nova\_virt\_type: kvm

**We then check the integrity of our configuration files by navigating to the directory** $ cd /opt/openstack-ansible/playbooks **and using the following script:** openstack-ansible setup-everything.yml --syntax-check

**We ran into a small problem when we first started running our playbooks and found that ansible was having a problem running sysstat. The correction needed was to modify** ansible.cfg **and adding the following line to the end of the file:**

scp\_if\_ssh = True

**Playbooks**

**Finally, we start running playbooks. The playbooks may fail or will have some warning about ignored problems. Usually if you simply rerun the playbook that had a problem, they will finish installing correctly. From the same** /opt/openstack-ansible/playbooks **directory we run the playbooks in this order:**

$ setup-hosts.yml

$ setup-infra.yml

$ setup-openstack.yml

**Our install hung on lxc and we manually restarted it by using the following commands:**

$ setup-haproxy.yml

$ setup-infrastructure.yml

$ memcached-install.yml

**We then listed the containers to verify they were created properly:**

$ lxc-ls -f

**Next, we connected to the containers:**

$ lxc-attach -n infra1\_neutron\_agents\_container-1564d34 **# (This last contiguous variable was a container listed after ansible created containers for us. The number at the end will probably be different for each creation)**

**Once all the playbooks have been run successfully and/or containers successfully deployed, we can use Horizon to open our cloud and spin up VM’s...**

**For reference, the complete interface file is here:**

<https://gist.github.com/Frank-ZhangXin/76a98d7f37c292f98827>

**Cisco switch setting:**

<http://www.cisco.com/c/en/us/td/docs/switches/datacenter/sw/5_x/nx-os/layer2/configuration/guide/Cisco_Nexus_7000_Series_NX-OS_Layer_2_Switching_Configuration_Guide_Release_5-x_chapter4.html#con_1273370>

<http://xmodulo.com/linux-tcpip-networking-net-tools-iproute2.html>

<http://www.dedoimedo.com/computers/dual-boot-windows-7-ubuntu.html>

**Neutron Networking Reference:**

<http://docs.openstack.org/liberty/networking-guide/scenario_legacy_lb.html>