OpenStack Cloud Deployment on Cisco UCS C200 M2 Servers

This Tech Note steps through setting up an OpenStack Cloud (Cactus release), comprising a cluster of compute and storage nodes each running Ubuntu 10.10. Each node is a Cisco UCS C200 M2 High-Density Rack-Mount Server. This document builds on installation instructions described in *OpenStack Compute and Storage Administration Guides*, but is not meant to supersede those documents.

Table of Contents

Table of Figures

Introduction

OpenStack is a collection of open source technologies that provide massively scalable open source cloud computing software. This Tech Note documents our experience in setting up an OpenStack Cloud (Cactus release), comprising a cluster of compute and storage nodes with each running Ubuntu 10.10). Each node is a Cisco UCS C200 M2 High-Density Rack-Mount Server. This document can be used as a reference for deploying a similar cluster. It builds on installation instructions described in *OpenStack Compute and Storage Administration Guides*[1], but is a more streamlined method that is specific to our deployment. We also attempt to provide additional details where the original documentation is short. We hope the reader finds our troubleshooting and workaround tips useful if problems develop during and after deployment. Please note that this document is not meant to supersede the official OpenStack installation and administration document. We encourage the reader to first consult that documentation to understand the OpenStack concepts and installation procedure.

Cisco UCS C200 M2 High-Density Rack-Mount Server[2]

The Cisco UCS C200 M2 Server is a high-density, 2-socket, 1 rack unit (RU) rack-mount server built for production-level network infrastructure, web services, and mainstream data center, branch, and remote-office applications. The configuration of each server used in our deployment is as follows:

- 2 x Intel(R) Xeon(R) CPU E5620 @ 2.40GHz
- 4 internal SAS or SATA disk drives; each 2 terabytes (TB)
- 24 GB of industry-standard double data rate (DDR3) main memory
- 4 Gigabit Ethernet ports

Cluster Topology

Our deployment consists of a cluster of four C200 servers. One server serves as the OpenStack *Cloud Controller*. The other three servers are configured as compute nodes. We recommend setting up the deployment such that the OpenStack management/control network is separate from the data network. (By management/control network, we imply the network which is used to access the servers, and on which the OpenStack processes exchange messages. By data network, we imply the network on which the virtual machines instantiated by OpenStack communicate with each other.) We leverage two network ports on each of these servers, such that one port is on the management/control network, and the other one is on the data network. Please note that the standard OpenStack installation process uses only one network for all communications. Figure 1 shows the topology.

[1] http://docs.openstack.org/
[2] http://www.cisco.com/en/US/products/ps10891/index.html

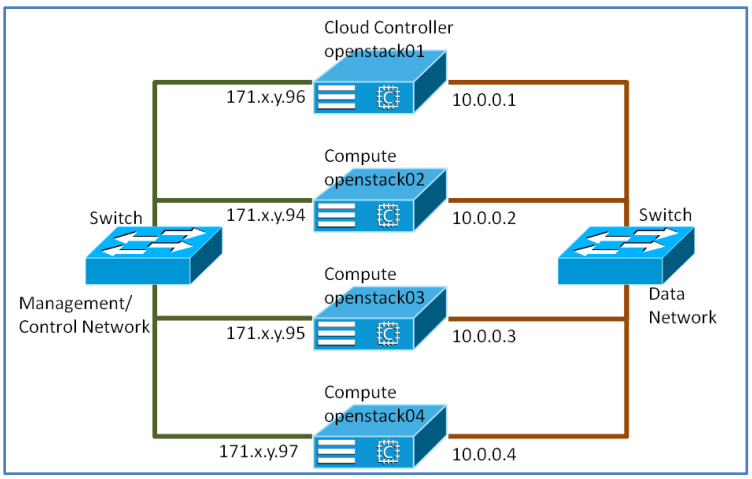

Figure 1: OpenStack Cloud Deployment on a C200 cluster[3]

OpenStack Compute Installation

The scripted installation works well for installing OpenStack, both on the Cloud Controller, and also on the other compute nodes. We will follow that approach for the installation. In our installation, we will run all the services on the Cloud Controller, and only the *nova-compute* service on the compute nodes. Note that in this set up, the Cloud Controller also serves as one of the compute nodes. We suggest this approach since you can get started running and testing virtual machine instances even with installing just the Cloud Controller, and adding one or more compute nodes later as required.

Installation on the Cloud Controller

Configuring the bridge

The virtual machine instances running on this node will communicate with the data network by connecting to a Linux bridge. We will first need to configure this bridge. We will use the eth1 port on our server for the data network (and we will configure it as a slave of br100). Our /etc/network/interfaces file looks like this:

```
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
auto eth0
iface eth0 inet static
address 171.x.y.96
gateway 171.x.y.1
netmask 255.255.254.0

auto br100
iface br100 inet static
        bridge_ports eth1
        bridge_stp off
        bridge_maxwait 0
        bridge_fd 0
```

[3] We have masked some of the digits on the 171. addresses used in this document with characters ‘x’ and ‘y’

Restart networking:

```
/etc/init.d/networking restart
```

An IP address will get automatically assigned to the bridge when we run the nova-network service. In this case, the first IP address in the range specified for the network will be used (i.e.; 10.0.0.1). Currently, there does not seem to be a way to configure this.

Running the Installation Script

Download the installation script:

```
wget --no-check-certificate https://github.com/dubsquared/OpenStack-NOVA-Installer-Script/raw/master/nova-CC-install-v1.1.sh
```

Ensure you can execute the script by modifying the permissions on the script file:

```
sudo chmod 755 nova-CC-install-v1.1.sh
```

Run the script with root permissions:

```
sudo ./nova-CC-install-v1.1.sh
```

You will be guided through the following prompts:

```
Step 1: Setting up the database.

mySQL User Config
##################

Desired mySQL Password:
Verify password:
```

Please enter a password for the “root” user on the MySQL database. Note this password as it might be required later during troubleshooting to access the MySQL database using MySQL client.

Next, you will be asked to enter the IP address for different services which run on the Cloud Controller.

```
S3 Host IP (Default is 10.0.0.3 -- ENTER to accept):171.x.y.96

RabbitMQ Host IP (Default is 10.0.0.3 -- ENTER to accept): 171.x.y.96

Cloud Controller Host IP (Default is 10.0.0.3 -- ENTER to accept): 171.x.y.96

mySQL Host IP (Default is 10.0.0.3 -- ENTER to accept): 171.x.y.96
```

Note that we have entered the IP address of the Cloud Controller on the management/controller network.

Next, you will be prompted for details about the *project* that will serve as the isolated resource container for your activities.

```
Nova project user name:<some_username>
Nova project name:test01
Desired network + CIDR for project (normally x.x.x.x/24):10.0.0.0/24
```

```
How many networks for project:1
How many availible IPs per project network:256
```

Make a note of the username and the project name that you enter here.

Currently only one network is supported per project.

Next you will be asked to enter details for the bridge configuration:

```
Please enter your local server IP (Default is 10.0.0.1 -- ENTER to accept):

Please enter your broadcast IP (Default is 10.0.0.255 -- ENTER to accept):

Please enter your netmask (Default is 255.255.255.0 -- ENTER to accept):

Please enter your gateway (Default is 171.x.y.1 -- ENTER to accept):10.0.0.1
```

We have used the IP addresses on the data network for this configuration. In our case, the defaults suggested by the script were correct (since we had already assigned an IP address to our br100 earlier). However, if you do not see these defaults, enter the appropriate IP address details as per the addressing scheme you have chosen for your data network.

Next, you will be prompted for the default name server.

```
Please enter your default nameserver (Default is 171.x.y.183 -- ENTER to accept):
```

The default is being suggested from your eth0 configuration. We accept that.

At this point, the script will start installing all the packages. Wait for it to complete successfully. If successful, the installation will also start all the services. Check by doing the following:

```
#ps -eaf | grep nova
root     31750 31742  0 07:42 ?        00:00:00 /usr/bin/python /usr/bin/nova-objectstore --uid
117 --gid 65534 --pidfile /var/run/nova/nova-objectstore.pid --flagfile=/etc/nova/nova.conf --
nodaemon --logfile=/var/log/nova/nova-objectstore.log
nova     32323     1  0 Apr19 ?        00:00:00 su -c nova-network --flagfile=/etc/nova/nova.conf
nova
nova     32340 32323  1 Apr19 ?        00:28:43 /usr/bin/python /usr/bin/nova-network --
flagfile=/etc/nova/nova.conf
nova     32393     1  0 Apr19 ?        00:00:00 su -c nova-compute --flagfile=/etc/nova/nova.conf
nova
nova     32410 32393  1 Apr19 ?        00:28:59 /usr/bin/python /usr/bin/nova-compute --
flagfile=/etc/nova/nova.conf
nova     32454     1  0 Apr19 ?        00:00:00 su -c nova-api --flagfile=/etc/nova/nova.conf
nova
nova     32489 32454  0 Apr19 ?        00:00:15 /usr/bin/python /usr/bin/nova-api --
flagfile=/etc/nova/nova.conf
nova     32501     1  0 Apr19 ?        00:00:00 su -c nova-scheduler --
flagfile=/etc/nova/nova.conf nova
nova     32508 32501  1 Apr19 ?        00:21:55 /usr/bin/python /usr/bin/nova-scheduler --
flagfile=/etc/nova/nova.conf
```

Post Script Installation

Once the installation has completed successfully, you will see that a /root/creds/novarc file has been created.

The novarc file will look like this:

```
NOVA_KEY_DIR=$(pushd $(dirname $BASH_SOURCE)>/dev/null; pwd; popd>/dev/null)
export EC2_ACCESS_KEY="<some_actual_key_here>:test01"
export EC2_SECRET_KEY="<some_actual_key_here>"
export EC2_URL="http://171.x.y.96:8773/services/Cloud"
export S3_URL="http://171.x.y.96:3333"
export EC2_USER_ID=42 # nova does not use user id, but bundling requires it
export EC2_PRIVATE_KEY=${NOVA_KEY_DIR}/pk.pem
export EC2_CERT=${NOVA_KEY_DIR}/cert.pem
export NOVA_CERT=${NOVA_KEY_DIR}/cacert.pem
export EUCALYPTUS_CERT=${NOVA_CERT} # euca-bundle-image seems to require this set
alias ec2-bundle-image="ec2-bundle-image --cert ${EC2_CERT} --privatekey ${EC2_PRIVATE_KEY} --
user 42 --ec2cert ${NOVA_CERT}"
alias ec2-upload-bundle="ec2-upload-bundle -a ${EC2_ACCESS_KEY} -s ${EC2_SECRET_KEY} --url
${S3_URL} --ec2cert ${NOVA_CERT}"
export NOVA_API_KEY="<some_actual_key_here>"
export NOVA_USERNAME="<username>"
export NOVA_URL=http://171.x.y.96:8774/v1.0/
```

Append the contents of this file to your profile file (eg: ~/.bashrc) and source it for this session.

```
cat /root/creds/novarc >> ~/.bashrc
source ~/.bashrc
```

You will also find some .pem files in the /root/creds/ directory. These .pem files have to be copied to the $NOVA_KEY_DIR path. (You will see these .pem files being referenced in the novarc file at that path.)

Create a “nova” group, so you can set permissions on the configuration file:

```
sudo addgroup nova
```

The nova.config file should have its owner set to root:nova, and mode set to 0640, since the file contains your MySQL server’s username and password.

```
chown -R root:nova /etc/nova
chmod 640 /etc/nova/nova.conf
```

These are the commands you run to ensure the database schema is current, and then set up a user and project:

```
/usr/bin/nova-manage db sync
/usr/bin/nova-manage user admin <user_name>
/usr/bin/nova-manage project create <project_name><user_name>
```

Note that we had earlier used the project name “test01”, so we would have used that here.

Network Configuration using FlatDHCPManager

Edit the /etc/nova/nova.conf to change the network manager to *FlatDHCPManager*. For our setup, the nova.conf looks like this:

```
--dhcpbridge_flagfile=/etc/nova/nova.conf
--dhcpbridge=/usr/bin/nova-dhcpbridge
--logdir=/var/log/nova
--state_path=/var/lib/nova
--lock_path=/var/lock/nova
--verbose
--s3_host=171.x.y.96
--rabbit_host=171.x.y.96
--cc_host=171.x.y.96
```

```
--ec2_url=http://171.x.y.96:8773/services/Cloud
--FAKE_subdomain=ec2
--routing_source_ip=171.x.y.96
--verbose
--sql_connection=mysql://root:password@171.x.y.96/nova
--network_manager=nova.network.manager.FlatDHCPManager
--network_size=256
--fixed_range=10.0.0.0/24
--flat_network_dhcp_start=10.0.0.11
```

Note that in the above configuration, we are indicating the VM instances should start getting allocated with IPs starting from 10.0.0.11, since we want to reserve IPs 10.0.0.0 (network), and 10.0.0.1 to 10.0.0.10 for the bridges on the Cloud Controller and one or more compute nodes. (However, configuring this file does not ensure that this configuration is correctly reflected in the DB. Instructions to ensure that are provided later in this section.)

Check the MySQL DB if a network entries has already been created during the scripted installation process.

```
mysql -uroot -p<password> nova -e 'select * from networks;'
```

If you see one or more entries, then do the following:

```
mysql -uroot -p<password> nova -e 'delete from networks where id > 0;'
mysql -uroot -p<password> nova -e 'delete from fixed_ips where id > 0;'
```

This will remove any previous network configuration from your DB.

Now create the network:

```
usr/bin/nova-manage network create 10.0.0.0/24 1 255
```

This should populate two tables in the DB, the *networks* table and the *fixed_ips* table.

Check the networks table:

```
# mysql -uroot -p<password> nova -e 'select * from networks;'
+---------------------+------------+------------+---------+----+----------+-------------+---------
-------+--------+----------+------------+------+------+--------------------+-----------------+---
------------------+------------+------------+-------------+---------+------------+-------+-------
-----+
| created_at          | updated_at | deleted_at | deleted | id | injected | cidr        | netmask
| bridge | gateway  | broadcast  | dns  | vlan | vpn_public_address | vpn_public_port |
vpn_private_address | dhcp_start | project_id | host        | cidr_v6 | gateway_v6 | label |
netmask_v6 |
+---------------------+------------+------------+---------+----+----------+-------------+---------
-------+--------+----------+------------+------+------+--------------------+-----------------+---
------------------+------------+------------+-------------+---------+------------+-------+-------
-----+
| 2011-04-08 20:19:18 | NULL       | NULL       |       0 |  1 |        0 | 10.0.0.0/24 |
255.255.255.0 | br100  | 10.0.0.1 | 10.0.0.255 | NULL | NULL | NULL               |
NULL | NULL               | 10.0.0.11  | test01     | openstack01 | NULL    | NULL       | NULL
| NULL       |
+---------------------+------------+------------+---------+----+----------+-------------+---------
-------+--------+----------+------------+------+------+--------------------+-----------------+---
------------------+------------+------------+-------------+---------+------------+-------+-------
-----+
```

Specifically, check that the *cidr, netmask, bridge, gateway, broadcast, dhcp_start, project_id*, and *host fields* are correctly populated.

If any of the fields are not correctly populated, set them by doing:

```
mysql -uroot -p<password> nova -e 'update networks set <column_name> = "<value>" where id =
<id_number>;'
```

After setting the fields, check that the changes have taken effect.

Next, you need to set reserve the IPs (10.0.0.0 to 10.0.0.10). This is done by setting the *reserved* field in the *fixed_ips* table for the relevant rows:

```
mysql -uroot -p<password> nova -e 'update fixed_ips set reserved = 1 where address = "10.0.0.1";'
```

Do this for all IP addresses you want to reserve.

After setting the fields, check that the changes have taken effect.

Stop any running instances of dnsmasq which are listening on the 10.0.0.1 address:

```
#ps -eaf | grep dns
nobody   24784     1  0 Apr17 ?        00:00:00 dnsmasq --strict-order --bind-interfaces --conf-
file= --domain=novalocal --pid-file=/var/lib/nova/networks/nova-br100.pid --listen-
address=10.0.0.1 --except-interface=lo --dhcp-range=10.0.0.11,static,120s --dhcp-
hostsfile=/var/lib/nova/networks/nova-br100.conf --dhcp-script=/usr/bin/nova-dhcpbridge --
leasefile-ro
root     24785 24784  0 Apr17 ?        00:00:00 dnsmasq --strict-order --bind-interfaces --conf-
file= --domain=novalocal --pid-file=/var/lib/nova/networks/nova-br100.pid --listen-
address=10.0.0.1 --except-interface=lo --dhcp-range=10.0.0.11,static,120s --dhcp-
hostsfile=/var/lib/nova/networks/nova-br100.conf --dhcp-script=/usr/bin/nova-dhcpbridge --
leasefile-ro
#kill -9 24784 24785
```

Restart all the services:

```
service libvirt-bin restart; service nova-network restart; service nova-compute restart; service
nova-api restart; service nova-objectstore restart; service nova-scheduler restart
```

Use the 'euca-authorize' command to enable ping and ssh access to all the VMs:

```
euca-authorize -P icmp -t -1:-1 default
euca-authorize -P tcp -p 22 default
```

If you want to use Ubuntu Enterprise Cloud images, you need the following iptables configuration so that UEC images can get metadata info:

```
#iptables -t nat -A PREROUTING -d 169.254.169.254/32 -p tcp -m tcp --dport 80 -j DNAT --to-
destination $NOVA_API_IP:8773
```

Bind this IP to the bridge interface so that any additional compute nodes in this cluster can get an arp response and resolve the 169.254.169.254 address to this machine:

```
ip addr add 169.254.169.254/32 dev br100
```

Testing the Installation by Publishing and Starting an Image

Once you have an installation, you want to get images that you can use in your Compute cloud. Download a sample Ubuntu image, and then use *uec-publish-tarball* to publish it:

```
image="ubuntu1010-UEC-localuser-image.tar.gz"
wget http://c0179148.cdn1.cloudfiles.rackspacecloud.com/ubuntu1010-UEC-localuser-image.tar.gz
uec-publish-tarball $image [bucket-name] [hardware-arch]
```

Here's an example of what this command looks like with data:

```
uec-publish-tarball ubuntu1010-UEC-localuser-image.tar.gz dub-bucket x86_64
```

The command in return should output three references: emi, eri and eki. You need to use the emi value (for example, “ami-zqkyh9th″) for the *euca-run-instances* command.

Now you can schedule, launch and connect to the instance, which you do with tools from the Euca2ools on the command line. Create the emi value from the *uec-publish-tarball* command, and then you can use the *euca-run-instances* command.

One thing to note here is once you publish the tarball, it has to untar before you can launch an image from it. Using the *euca-describe-images* command, wait until the state turns to "available" from "untarring."

Depending on the image that you're using, you need a public key to connect to it. Some images have built-in accounts already created. Images can be shared by many users, so it is dangerous to put passwords into the images. Nova therefore supports injecting ssh keys into instances before they are booted. This allows a user to login to the instances that he or she creates securely. Generally, the first thing that a user does when using the system is to create a key pair. Key pairs provide secure authentication to your instances. As part of the first boot of a virtual image, the private key of your key pair is added to root’s authorized_keys file. Nova generates a public and private key pair, and sends the private key to the user. The public key is stored so that it can be injected into instances.

Key pairs are created through the api and you use them as a parameter when launching an instance. They can be created on the command line using the euca2ools script *euca-add-keypair*. Refer to the main page for the available options. Example usage:

```
euca-add-keypair test > test.pem
chmod 600 test.pem
```

Now, you can run the instances:

```
euca-run-instances -k test -t m1.tiny ami-zqkyh9th
```

Here's a description of the parameters used above:
-t what type of image to create
-k name of the key to inject in to the image at launch

Optionally, you can use the -n parameter to indicate how many images of this type to launch.

Check the status of the instance by using the *euca-describe-instances* command. The instance will go from “launching” to “running” in a short time, and you should be able to connect via SSH using the 'ubuntu' account, with the password 'ubuntu': (replace $ipaddress with the one you got from euca-describe-instances):

```
ssh ubuntu@$ipaddress
```

Installing Compute Nodes

Once the Cloud Controller is installed successfully you can add more compute nodes to the cluster. (Note that if you have followed the instructions above for the installation, you have already installed one compute node on the Cloud Controller itself.)

As indicated in the OpenStack Compute Administration guide, there are many different ways in which just the OpenStack compute service can be installed on the nodes other than the Cloud Controller. One of the simpler approaches is to run the installation script like we did on the Cloud Controller. This will install some additional packages than what is required just for the nova-compute.

Configuring the Bridge

The virtual machine instances running on this node will communicate with the data network by connecting to a Linux bridge. We will first need to configure this bridge. We will use the eth1 port on our server for the data network (and we will configure it as a slave of br100). Our /etc/network/interfaces file looks like this (Note that in this case, we are also setting the IP address of the bridge manually):

```
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
auto eth0
iface eth0 inet static
address 171.x.y.96
gateway 171.x.y.1
netmask 255.255.254.0

auto br100
iface br100 inet static
        bridge_ports eth1
        bridge_stp off
        bridge_maxwait 0
        bridge_fd 0
address 10.0.0.2
       netmask 255.255.255.0
        broadcast 10.0.0.255
gateway 10.0.0.1
```

Restart networking:

```
/etc/init.d/networking restart
```

Running the Installation Script

Download the installation script:

```
wget --no-check-certificate https://github.com/dubsquared/OpenStack-NOVA-Installer-Script/raw/master/nova-CC-install-v1.1.sh
```

Ensure you can execute the script by modifying the permissions on the script file:

```
sudo chmod 755 nova-CC-install-v1.1.sh
```

Run the script with root permissions:

```
sudo ./nova-CC-install-v1.1.sh
```

Enter the same values, as in section 0 until you reach the bridge configuration. In the bridge configuration, enter the IP address of the bridge on this machine.

Once the installation script completes successfully, stop all OpenStack services which were started by the installation script. (We will restart them later after completing the configuration on this node.):

```
service libvirt-bin stop; service nova-network stop; service nova-compute stop; service nova-api stop; service nova-objectstore stop; service nova-scheduler stop
```

Post Script Installation

Copy the contents of the /root/creds/novarc file on the Cloud Controller, and append them to your profile file (eg: ~/.bashrc). Source the .bashrc for this session:

```
source ~/.bashrc
```

Copy the *.pem files from the /root/creds/ directory on the Cloud Controller to the $NOVA_KEY_DIR path on this node.

Create a “nova” group, so you can set permissions on the configuration file:

```
sudo addgroup nova
```

The nova.config file should have its owner set to root:nova, and mode set to 0640, since the file contains your MySQL server’s username and password:

```
chown -R root:nova /etc/nova
chmod 640 /etc/nova/nova.conf
```

Replace the contents of the /etc/nova/nova.conf file on this node with the contents of the /etc/nova/nova.conf file on the Cloud Controller.

Restart the nova-compute service:

```
service nova-compute restart
```

Testing the Installation on this Node

On the Cloud Controller, check the MySQL DB by running:

```
mysql -uroot -p<password> nova -e 'select * from services;'
```

You should see that *nova-compute* is also running on this freshly installed host. (Also, check that this is the only service listed running on this node.)

As described in section 0, start another VM instance from the Cloud Controller:

```
euca-run-instances -k test -t m1.tiny ami-zqkyh9th
```

The *nova-scheduler* service decides on which node to instantiate the new VM. Running euca-describe-instances will tell you if the VM is running on this freshly installed compute node.

If you see the VM running on this node, try to ssh to it. If ssh is successful, the installation and configuration has worked. You now have an OpenStack cluster working.

OpenStack Dashboard Installation

The OpenStack dashboard in this deployment in installed on the Cloud Controller. The instructions provided in the *OpenStack Compute Administration Guide* should be followed to perform this installation. You might experience some problems with installing certain packages due to time out issues. Please refer to the troubleshooting section of this document to see how you can workaround those problems.

Shown below is a snapshot of the dashboard in our deployment after the user is logged in. At this instant when the snapshot was taken, four VM instances were running across this cluster.

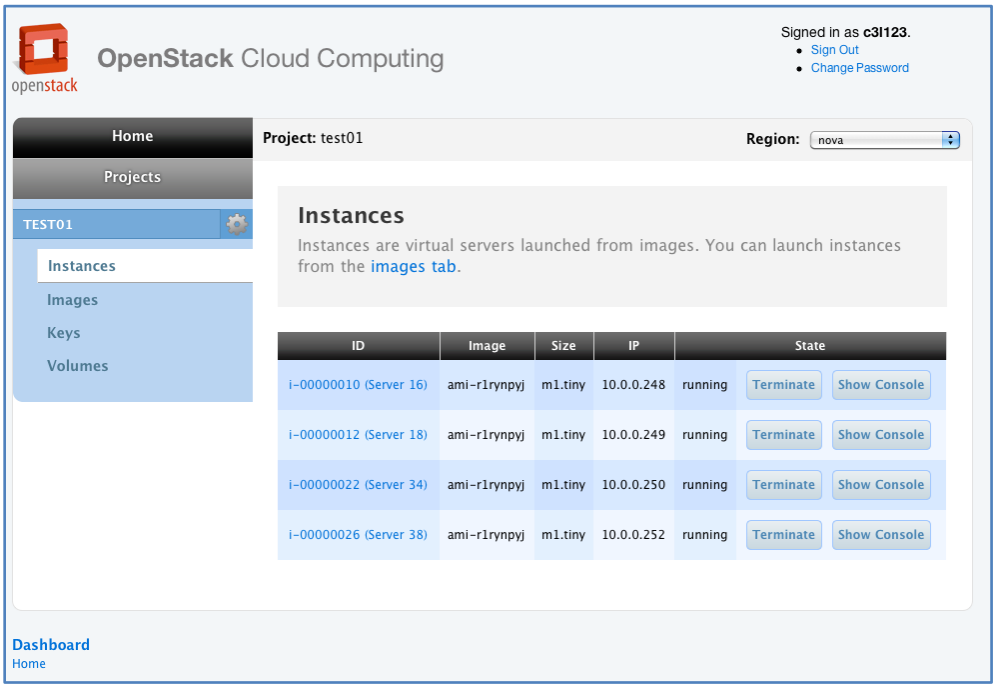

Figure 2: OpenStack Dashboard

OpenStack Storage Installation

Our deployment consists of a cluster of four C200 Servers. One server serves as the OpenStack *Proxy* and *Auth* node. The other three servers are configured as *Storage* nodes. As before, we recommend setting up the deployment such that the OpenStack management/control network is separate from the data network. (By management/control network, we imply the network which is used to access the servers, and on which the OpenStack processes exchange messages. By data network, we imply the network on which the virtual machines instantiated by OpenStack communicate with each other.) The figure below shows the topology.

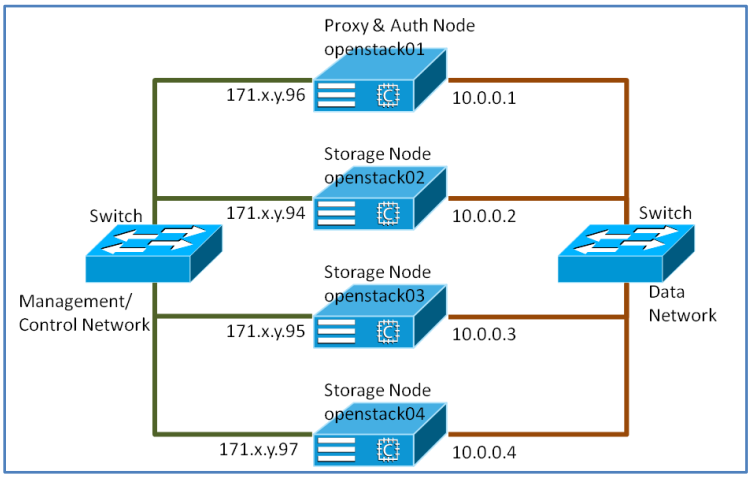

Figure 3: OpenStack Storage Deployment on a C200 cluster

In the following sub-sections, we describe how OpenStack Object Storage can be installed on the above cluster. Most of these instructions follow the process described in the OpenStack Storage Administration Guide. However, there are areas where we provide more details and others where we have corrected discrepancies in the OpenStack guide. Keeping the reader's convenience in mind, we outline the installation process in its entirety here.

Install and Configure the Packages

Install common OpenStack Object Storage software and pre-requisites:

```
apt-get install python-software-properties
add-apt-repository ppa:swift-core/ppa
apt-get update
apt-get install swift openssh-server
```

Create and populate configuration directories on all nodes:

```
mkdir -p /etc/swift
chown -R swift:swift /etc/swift/
```

Create /etc/swift/swift.conf:

```
[swift-hash]
# random unique string that can never change (DO NOT LOSE)
swift_hash_path_suffix = <changeme>
```

It is assumed that all commands are run as the root user.

Install Swift-proxy Service

On the Proxy node,

```
apt-get install swift-proxy memcached
```

Create self-signed cert for SSL:

```
cd /etc/swift
openssl req -new -x509 -nodes -out cert.crt -keyout cert.key
```

Modify memcached to listen on the default interfaces. Preferably, this should be on a local, non-public network. Edit the following line in /etc/memcached.conf, changing:

```
-l 127.0.0.1
to
-l <10.0.0.1>
```

Restart the memcached server:

```
service memcached restart
```

Create /etc/swift/proxy-server.conf:

```
[DEFAULT]
cert_file = /etc/swift/cert.crt
key_file = /etc/swift/cert.key
bind_port = 8080
workers = 8
user = swift

[pipeline:main]
pipeline = healthcheck cache auth proxy-server

[app:proxy-server]
use = egg:swift#proxy
allow_account_management = true

[filter:auth]
use = egg:swift#auth
ssl = true

[filter:healthcheck]
use = egg:swift#healthcheck

[filter:cache]
use = egg:swift#memcache
memcache_servers = <10.0.0.1>:11211
```

Create the Account, Container and Object Rings:

On the Proxy node:

```
cd /etc/swift
swift-ring-builder account.builder create 18 3 1
swift-ring-builder container.builder create 18 3 1
swift-ring-builder object.builder create 18 3 1
```

For every storage device, on each node, add entries to each ring:

```
swift-ring-builder account.builder add z1-10.0.0.2:6002/sdd1 100
swift-ring-builder container.builder add z1-10.0.0.2:6001/sdd1 100
```

```
swift-ring-builder object.builder add z1-10.0.0.2:6000/sdd1 100
swift-ring-builder account.builder add z2-10.0.0.3:6002/sdd1 100
swift-ring-builder container.builder add z2-10.0.0.3:6001/sdd1 100
swift-ring-builder object.builder add z2-10.0.0.3:6000/sdd1 100
swift-ring-builder account.builder add z3-10.0.0.4:6002/sdd1 100
swift-ring-builder container.builder add z3-10.0.0.4:6001/sdd1 100
swift-ring-builder object.builder add z3-10.0.0.4:6000/sdd1 100
```

In our case, there are three zones, z1(10.0.0.2), z2(10.0.0.3), and z3(10.0.04).

Verify the ring contents for each ring:

```
swift-ring-builder account.builder
swift-ring-builder container.builder
swift-ring-builder object.builder
```

Rebalance the rings:

```
swift-ring-builder account.builder rebalance
swift-ring-builder container.builder rebalance
swift-ring-builder object.builder rebalance
```

Rebalancing rings can take some time.

Copy the account.ring.gz, container.ring.gz, and object.ring.gz files to each of the *Storage* nodes in /etc/swift.

Make sure all the config files are owned by the swift user:

```
chown -R swift:swift /etc/swift
```

Start Proxy services:

```
swift-init proxy start
```

Installing and Configuring the Auth Node

On the Proxy node:

Install swift-auth service:

```
apt-get install swift-auth
```

Create /etc/swift/auth-server.conf:

```
[DEFAULT]
cert_file = /etc/swift/cert.crt
key_file = /etc/swift/cert.key
user = swift

[pipeline:main]
pipeline = auth-server

[app:auth-server]
use = egg:swift#auth
default_cluster_url = https://<openstack01>:8080/v1
# Highly recommended to change this key to something else!
super_admin_key = devauth
Start Auth services:

swift-init auth start
```

```
chown swift:swift /etc/swift/auth.db
swift-init auth restart           # 1.1.0 workaround because swift creates auth.db owned as root
```

Installing and Configuring the Storage Nodes

OpenStack Object Storage should work on any modern file system that supports Extended Attributes (XATTRS). We currently recommend XFS as it demonstrated the best overall performance for the swift use case after considerable testing and benchmarking at Rackspace. It is also the only filesystem that has been thoroughly tested.

Install Storage Node Packages

On each of the three *Storage* nodes, perform all the steps in the sub-section:

```
apt-get install swift-account swift-container swift-object xfsprogs
```

Set up the XFS volume on the storage device (follow the commands and input in the following snippet):

```
root@openstack03:~# fdisk /dev/sdd
Device contains neither a valid DOS partition table, nor Sun, SGI or OSF disklabel
Building a new DOS disklabel with disk identifier 0x45a10b80.
Changes will remain in memory only, until you decide to write them.
After that, of course, the previous content won't be recoverable.

Warning: invalid flag 0x0000 of partition table 4 will be corrected by w(rite)

WARNING: DOS-compatible mode is deprecated. It's strongly recommended to
         switch off the mode (command 'c') and change display units to
         sectors (command 'u').

Command (m for help): n
Command action
   e   extended
   p   primary partition (1-4)
p
Partition number (1-4): 1
First cylinder (1-243201, default 1):
Using default value 1
Last cylinder, +cylinders or +size{K,M,G} (1-243201, default 243201):
Using default value 243201

Command (m for help): p

Disk /dev/sdd: 2000.4 GB, 2000398934016 bytes
255 heads, 63 sectors/track, 243201 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x45a10b80

   Device Boot      Start         End      Blocks   Id  System
/dev/sdd1               1      243201  1953512001   83  Linux

Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table.
Syncing disks.

root@openstack03:~# partprobe

root@openstack03:~# fdisk -l
```

```
WARNING: GPT (GUID Partition Table) detected on '/dev/sdc'! The util fdisk doesn't support GPT.
Use GNU Parted.


Disk /dev/sdc: 2000.4 GB, 2000398934016 bytes
255 heads, 63 sectors/track, 243201 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x00000000

   Device Boot      Start         End      Blocks   Id  System
/dev/sdc1               1      243202  1953514583+  ee  GPT

Disk /dev/sdd: 2000.4 GB, 2000398934016 bytes
255 heads, 63 sectors/track, 243201 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x45a10b80

   Device Boot      Start         End      Blocks   Id  System
/dev/sdd1               1      243201  1953512001   83  Linux

Disk /dev/sde: 2000.4 GB, 2000398934016 bytes
255 heads, 63 sectors/track, 243201 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0xbbbab9b8

Disk /dev/sde doesn't contain a valid partition table

Disk /dev/sdf: 2000.4 GB, 2000398934016 bytes
255 heads, 63 sectors/track, 243201 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0xbbbab9b8

Disk /dev/sdf doesn't contain a valid partition table

root@openstack03:~# cat /proc/partitions
major minor  #blocks  name

   8       32 1953514584 sdc
   8       33       1024 sdc1
   8       34 1881237504 sdc2
   8       35   72274944 sdc3
   8       48 1953514584 sdd
   8       49 1953512001 sdd1
   8       64 1953514584 sde
   8       80 1953514584 sdf

root@openstack03:~# mkfs.xfs -i size=1024 /dev/sdd1
meta-data=/dev/sdd1              isize=1024   agcount=4, agsize=122094500 blks
         =                       sectsz=512   attr=2
data     =                       bsize=4096   blocks=488378000, imaxpct=5
         =                       sunit=0      swidth=0 blks
naming   =version 2              bsize=4096   ascii-ci=0
log      =internal log           bsize=4096   blocks=238465, version=2
         =                       sectsz=512   sunit=0 blks, lazy-count=1
realtime =none                   extsz=4096   blocks=0, rtextents=0
root@openstack03:~# echo "/dev/sdd1 /srv/node/sdd1 xfs noatime,nodiratime,nobarrier,logbufs=8 0
0" >> /etc/fstab
root@openstack03:~# mkdir -p /srv/node/sdd1
root@openstack03:~# mount /srv/node/sdd1
root@openstack03:~# chown -R swift:swift /srv/node
```

Create /etc/rsyncd.conf:

```
uid = swift
gid = swift
```

```
log file = /var/log/rsyncd.log
pid file = /var/run/rsyncd.pid
address = <STORAGE_LOCAL_NET_IP, either one of 10.0.0.2/3/4>

[account]
max connections = 2
path = /srv/node/
read only = false
lock file = /var/lock/account.lock

[container]
max connections = 2
path = /srv/node/
read only = false
lock file = /var/lock/container.lock

[object]
max connections = 2
path = /srv/node/
read only = false
lock file = /var/lock/object.lock
```

Edit the following line in /etc/default/rsync:

```
RSYNC_ENABLE = true
```

Start rsync daemon:

```
service rsync start
```

Create /etc/swift/account-server.conf:

```
[DEFAULT]
bind_ip = <STORAGE_LOCAL_NET_IP, either one of 10.0.0.2/3/4>
workers = 2

[pipeline:main]
pipeline = account-server
mount_check = false

[account-server]
use = egg:swift#account

[account-replicator]

[account-auditor]

[account-reaper]
```

Create /etc/swift/container-server.conf:

```
[DEFAULT]
bind_ip = <STORAGE_LOCAL_NET_IP, either one of 10.0.0.2/3/4>
workers = 2

[pipeline:main]
pipeline = container-server

[container-server]
use = egg:swift#container

[container-replicator]

[container-updater]

[container-auditor]
```

Create /etc/swift/object-server.conf:

```
[DEFAULT]
bind_ip = <STORAGE_LOCAL_NET_IP, either one of 10.0.0.2/3/4>
workers = 2

[pipeline:main]
pipeline = object-server

[object-server]
use = egg:swift#object

[object-replicator]

[object-updater]

[object-auditor]
```

Start the storage services:

```
swift-init object-server start
swift-init object-replicator start
swift-init object-updater start
swift-init object-auditor start
swift-init container-server start
swift-init container-replicator start
swift-init container-updater start
swift-init container-auditor start
swift-init account-server start
swift-init account-replicator start
swift-init account-auditor start
```

Check that the storage services have indeed started:

```
ps -eaf | grep swift
swift    13873     1  0 Apr15 ?        00:00:00 /usr/bin/python /usr/bin/swift-account-auditor
/etcswift/account-server.conf
swift    13874     1  0 Apr15 ?        00:00:00 /usr/bin/python /usr/bin/swift-account-server
/etc/swift/account-server.conf
swift    13875     1  0 Apr15 ?        00:00:02 /usr/bin/python /usr/bin/swift-container-auditor
/etc/swift/container-server.conf
swift    13876     1  0 Apr15 ?        00:01:35 /usr/bin/python /usr/bin/swift-container-
replicator /etc/swift/container-server.conf
swift    13877     1  0 Apr15 ?        00:00:00 /usr/bin/python /usr/bin/swift-container-server
/etc/swift/container-server.conf
swift    13878     1  0 Apr15 ?        00:00:04 /usr/bin/python /usr/bin/swift-container-updater
/etc/swift/container-server.conf
swift    13879     1  0 Apr15 ?        00:00:00 /usr/bin/python /usr/bin/swift-object-auditor
/etc/swift/object-server.conf
swift    13880     1  0 Apr15 ?        00:00:00 /usr/bin/python /usr/bin/swift-object-server
/etc/swift/object-server.conf
swift    13881     1  0 Apr15 ?        00:05:57 /usr/bin/python /usr/bin/swift-object-replicator
/etc/swift/object-server.conf
swift    13882     1  0 Apr15 ?        00:00:02 /usr/bin/python /usr/bin/swift-object-updater
/etc/swift/object-server.conf
swift    13924 13880  0 Apr15 ?        00:00:40 /usr/bin/python /usr/bin/swift-object-server
/etc/swift/object-server.conf
swift    13925 13880  0 Apr15 ?        00:00:40 /usr/bin/python /usr/bin/swift-object-server
/etc/swift/object-server.conf
swift    13931 13877  0 Apr15 ?        00:00:54 /usr/bin/python /usr/bin/swift-container-server
/etc/swift/container-server.conf
swift    13932 13877  0 Apr15 ?        00:00:53 /usr/bin/python /usr/bin/swift-container-server
/etc/swift/container-server.conf
swift    13934 13874  0 Apr15 ?        00:00:01 /usr/bin/python /usr/bin/swift-account-server
/etc/swift/account-server.conf
swift    13935 13874  0 Apr15 ?        00:00:01 /usr/bin/python /usr/bin/swift-account-server
/etc/swift/account-server.conf
```

If you do not see that all the services have started, please refer to the troubleshooting section to debug this issue.

Create OpenStack Object Storage admin Account and Verify the Installation

Run these commands from the *Proxy* node (which also runs the *Auth* service).

Create a user with administrative privileges (account = system, username = root, password = testpass). Make sure to replace devauth in the swift-auth-add-user command below with whatever super_admin key you assigned in the auth-server.conf file above. None of the values of account, username, or password are special. They can be anything.

```
swift-auth-add-user -K devauth -a system root testpass
```

Get an X-Storage-Url and X-Auth-Token:

```
curl -k -v -H 'X-Storage-User: system:root' -H 'X-Storage-Pass: testpass'
https://<openstack01>:11000/v1.0
```

Check that you can HEAD the account:

```
curl -k -v -H 'X-Auth-Token: <token-from-x-auth-token-above>' <url-from-x-storage-url-above>
```

Check that *st* works:

```
st -A https://<AUTH_HOSTNAME>:11000/v1.0 -U system:root -K testpass stat
```

Use *st* to upload a few files named ‘text[1-2].txt’ to a container named ‘myfiles’:

```
st -A https://<openstack01>:11000/v1.0 -U system:root -K testpass upload myfiles test1.txt
st -A https://<openstack01>:11000/v1.0 -U system:root -K testpass upload myfiles test2.txt
```

Use *st* to download all files from the ‘myfiles’ container:

```
st -A https://<openstack01>:11000/v1.0 -U system:root -K testpass download myfiles
```

If you are able to successfully retrieve the files test1.txt and test2.txt along with their contents, you have successfully configured OpenStack Object Storage.

Troubleshooting Tips

Compute

Not Able to Pull the Latest Cactus Release

If you are running the scripted installation, you will not run into this problem. If you are doing a manual installation based on the instructions provided in the OpenStack Compute Administration Guide, you might see this problem. This is because the following instruction

```
sudo add-apt-repository ppa:nova-core/trunk
```

points to the older repository.

In case you have already run the above command, you need to remove this repository from your repository list:

```
rm -rf /etc/apt/sources.list.d/nova-core-trunk-maverick.list
rm -rf /etc/apt/sources.list.d/nova-core-trunk-maverick.list.save
```

And, add the correct repository:

```
sudo add-apt-repository ppa:nova-core/release
```

Not Able to Upgrade from Bexar to Cactus

This is similar to the earlier issue. If you already have an earlier installation and you are trying to upgrade your installation from Bexar to Cactus by doing:

```
sudo apt-get dist-upgrade
```

you will still get the Bexar release. To get the Cactus release, you need to point to a different repository. Follow the instructions in the earlier sub-section to do this, then run the upgrade as above. After running the upgrade, sync the DB:

```
/usr/bin/nova-manage db sync
```

How to Create a New Network (and Delete the Existing One)

If you already have a network created and you want to create a new one, you will first need to remove the earlier network. The only way to do this is by cleaning the relevant entries in the DB. Assuming that you are not interested in keeping the currently running instances, you can do the following to clean the DB:

```
mysql -uroot -pc3l123 nova -e 'delete from instances where id > 0;'
mysql -uroot -pc3l123 nova -e 'delete from security_group_instance_association where id > 0;'
mysql -uroot -pc3l123 nova -e 'delete from fixed_ips where id > 0;'
mysql -uroot -pc3l123 nova -e 'delete from networks where id > 0;'
```

After cleaning the DB, follow the instructions given earlier on how to create a new network.

Not Able to Publish an Image (Getting an Invalid Cert Error)

When you run:

```
# uec-publish-tarball ubuntu1010-UEC-localuser-image.tar.gz dub-bucket x86_64
```

if you see the following error,

```
Tue Apr 19 00:27:23 PDT 2011: ====== extracting image ======
Warning: no ramdisk found, assuming '--ramdisk none'
kernel : maverick-server-uec-amd64-vmlinuz-virtual
ramdisk: none
image  : maverick-server-uec-amd64.img
Tue Apr 19 00:27:46 PDT 2011: ====== bundle/upload kernel ======
failed to bundle kernel maverick-server-uec-amd64-vmlinuz-virtual
failed: euca-bundle-image --destination /tmp/uec-publish-image.Qi26aw --arch i386 --image
/tmp/uec-publish-image.Qi26aw/.rename.zoWGb4/maverick-server-uec-amd64-vmlinuz-virtual --kernel
true
i386
Invalid certfailed to upload kernel
```

it is most likely because you do not have the /root/creds/*.pem files in the $NOVA_KEY_DIR path. Do the following and check if you see the *.pem files:

```
ls -ltr $NOVA_KEY_DIR/*.pem
```

If you don't see these files, copy them to this directory.

Running Instance Hangs in the "Scheduling" State

If you do an *euca-describe-instances* and see that the newly created instance is stuck in the "scheduling" state, it is most likely because the *nova-scheduler* process in not running on your Cloud Controller. Check:

```
ps -eaf | grep nova-scheduler
```

and, if you do not see it running, do:

```
service nova-scheduler start
```

UEC Image Instance Can Be Pinged, But Cannot Ssh

As a part of the booting process, the image tries to contact a metadata service to get the keys for this instance. If the instance is not able to reach this metadata service, the boot up process is not complete. Hence, you cannot ssh to this virtual machine instance. To see if the boot up process has completed, you need to check the console output of the instance.

You can get the console output of the instance by running:

```
euca-get-console-output <instance id as seen in euca-describe-images>
```

If you see the following lines in the console output:

```
[    0.507752] Freeing unused kernel memory: 308k freed^M
[    0.509535] Freeing unused kernel memory: 1612k freed^M
init: plymouth main process (49) killed by SEGV signal^M
init: plymouth-splash main process (281) terminated with status 2^M cloud-init start-local
running: Mon, 11 Apr 2011 09:49:42 +0000. up 1.52 seconds no instance data found in start-local
init: cloud-init-local main process (270) terminated with status 1^M cloud-init start running:
Mon, 11 Apr 2011 09:49:42 +0000. up 1.77 seconds
2011-04-11 09:49:42,989 - DataSourceEc2.py[WARNING]: waiting for metadata service at
http://169.254.169.254/2009-04-04/meta-data/instance-id

2011-04-11 09:49:42,990 - DataSourceEc2.py[WARNING]:   09:49:42 [ 1/100]: http error [500]

2011-04-11 09:49:44,031 - DataSourceEc2.py[WARNING]:   09:49:44 [ 2/100]: http error [500]
```

the likely problem is that you are not able reach the metadata service at 169.254.169.254. Make sure that you are able to ping the address 169.254.169.254 from the compute node on which this virtual machine instance is running. If your ping is not successful, you are most probably missing the following iptables configuration on the Cloud Controller:

```
#iptables -t nat -A PREROUTING -d 169.254.169.254/32 -p tcp -m tcp --dport 80 -j DNAT --to-
destination $NOVA_API_IP:8773
```

Also, bind this IP to the bridge interface so that any additional compute nodes in this cluster can get an arp response and resolve the 169.254.169.254 address to this machine:

```
ip addr add 169.254.169.254/32 dev br100
```

Socket Time Out Error During Dashboard Installation

While running the command:

```
python tools/install_venv.py ../../django-nova/trunk
```

if you see the following message:

```
Downloading/unpacking Django==1.2.3 (from -r /home/c3l123/openstack/src/openstack-
dashboard/trunk/openstack-dashboard/tools/pip-requires (line 3))
  Downloading Django-1.2.3.tar.gz (6.3Mb): 1.2Mb downloaded
Exception:
Traceback (most recent call last):
  File "/home/c3l123/openstack/src/openstack-dashboard/trunk/openstack-dashboard/.dashboard-
venv/lib/python2.6/site-packages/pip-1.0-py2.6.egg/pip/basecommand.py", line 126, in main
    self.run(options, args)
  File "/home/c3l123/openstack/src/openstack-dashboard/trunk/openstack-dashboard/.dashboard-
venv/lib/python2.6/site-packages/pip-1.0-py2.6.egg/pip/commands/install.py", line 223, in run
    requirement_set.prepare_files(finder, force_root_egg_info=self.bundle, bundle=self.bundle)
  File "/home/c3l123/openstack/src/openstack-dashboard/trunk/openstack-dashboard/.dashboard-
venv/lib/python2.6/site-packages/pip-1.0-py2.6.egg/pip/req.py", line 955, in prepare_files
    self.unpack_url(url, location, self.is_download)
  File "/home/c3l123/openstack/src/openstack-dashboard/trunk/openstack-dashboard/.dashboard-
venv/lib/python2.6/site-packages/pip-1.0-py2.6.egg/pip/req.py", line 1072, in unpack_url
    return unpack_http_url(link, location, self.download_cache, only_download)
  File "/home/c3l123/openstack/src/openstack-dashboard/trunk/openstack-dashboard/.dashboard-
venv/lib/python2.6/site-packages/pip-1.0-py2.6.egg/pip/download.py", line 441, in unpack_http_url
    download_hash = _download_url(resp, link, temp_location)
  File "/home/c3l123/openstack/src/openstack-dashboard/trunk/openstack-dashboard/.dashboard-
venv/lib/python2.6/site-packages/pip-1.0-py2.6.egg/pip/download.py", line 366, in _download_url
    chunk = resp.read(4096)
  File "/usr/lib/python2.6/socket.py", line 377, in read
    data = self._sock.recv(left)
  File "/usr/lib/python2.6/httplib.py", line 542, in read
    s = self.fp.read(amt)
  File "/usr/lib/python2.6/socket.py", line 377, in read
    data = self._sock.recv(left)
timeout: timed out

Storing complete log in /home/c3l123/.pip/pip.log
Command "/home/c3l123/openstack/src/openstack-dashboard/trunk/openstack-
dashboard/tools/with_venv.sh pip install -E /home/c3l123/openstack/src/openstack-
dashboard/trunk/openstack-dashboard/.dashboard-venv -r /home/c3l123/openstack/src/openstack-
dashboard/trunk/openstack-dashboard/tools/pip-requires" failed.
None
```

you need to first independently install that package in the appropriate virtualenv:

```
cd /home/c3l123/openstack/src/openstack-dashboard/trunk/openstack-dashboard/.dashboard-venv

source bin/activate

wget -O Django-1.2.3.tar.gz http://www.djangoproject.com/download/1.2.3/tarball/

pip install -E /home/c3l123/openstack/src/openstack-dashboard/trunk/openstack-

dashboard/.dashboard-venv Django-1.2.3.tar.gz

deactivate
```

and then retry running the earlier command.

Storage

Storage Services Do Not Start on the Storage Node

The *stdout* stream of the *swift-init* scripts is directed to /dev/null. So, you will not know if a particular service errors out and dies immediately after you start it.

In case you are not able to see any of your services running, you need to try to start that service individually, and check the output. For instance, there was an error in our container-server.conf file, and to debug it, we run that server individually:

```
root@openstack02:/etc/swift# swift-container-server /etc/swift/container-server.conf
Traceback (most recent call last):
  File "/usr/bin/swift-container-server", line 25, in <module>
    if not c.read(sys.argv[1]):
  File "/usr/lib/python2.6/ConfigParser.py", line 286, in read
    self._read(fp, filename)
  File "/usr/lib/python2.6/ConfigParser.py", line 510, in _read
    raise e
ConfigParser.ParsingError: File contains parsing errors: /etc/swift/container-server.conf
      [line 17]: '~                  \n'
```

Unable to Start the Account Server on the Storage Node

As described in the previous sub-section, start the account server individually, and check if you see the error as below:

```
root@openstack04:/etc/swift# swift-account-server /etc/swift/account-server.conf
Traceback (most recent call last):
  File "/usr/bin/swift-account-server", line 28, in <module>
    conf = dict(c.items('account-server'))
  File "/usr/lib/python2.6/ConfigParser.py", line 565, in items
    raise NoSectionError(section)
ConfigParser.NoSectionError: No section: 'account-server'
```

If you do see the above trace, then there is an error in the account-server.conf file. If you copied the contents of this file from the OpenStack Storage Administration Guide, you will see the strings “app:” prefixed to the string “account-server”. Apparently, the prefix should not be present. We have provided the corrected config files in the earlier installation section of this document.

The same fix applies to the configuration files of the *Container* and *Object* servers as well.