Deploying Ceph Storage Cluster + Calamari (for Ubuntu Server 16.04 LTS)

This documentation is aimed at System Administrators for deploying a Ceph Storage Cluster along with Calamari web interface. This can be used as an ideal setup for a small office environment, or if properly scaled up can serve as a backbone for an enterprise environment.

The contents of documentation are as follows.

Introduction to Ceph

Ceph is based on RADOS or Reliable Autonomic Distributed Object Store.
RADOS distributes objects across the storage cluster and replicates objects for fault tolerance. It consists of the following major components.

1) OSD [Object Storage Device] daemon
2) MDS [Meta-Data Server]
3) MON [Monitor]

1) OSD [Object Storage Device] daemon

Ceph OSD daemon stores data, handles data replication, recovery, backfilling, rebalancing, and provides some monitoring information to Ceph Monitors by checking other Ceph OSD daemons for a heartbeat. A Ceph Storage Cluster requires at least two Ceph OSD daemons to achieve an active plus clean state when the cluster makes two copies of our data.


Ceph makes 3 copies by default, but we can adjust it

2) MDS [Meta-Data Server]

Ceph MDS stores metadata on behalf of the Ceph Filesystem. Ceph Metadata Servers make it feasible for POSIX file system users to execute basic commands like ls, find, etc. without placing an enormous burden on the Ceph Storage Cluster.

Ceph stores a client's data as objects within storage pools. Using the CRUSH(Controlled Replication Under Scalable Hashing)[1][2] algorithm, Ceph calculates which placement group should contain the object, and further calculates which Ceph OSD Daemon should store the placement group. The CRUSH algorithm enables the Ceph Storage Cluster to scale, rebalance, and recover dynamically.

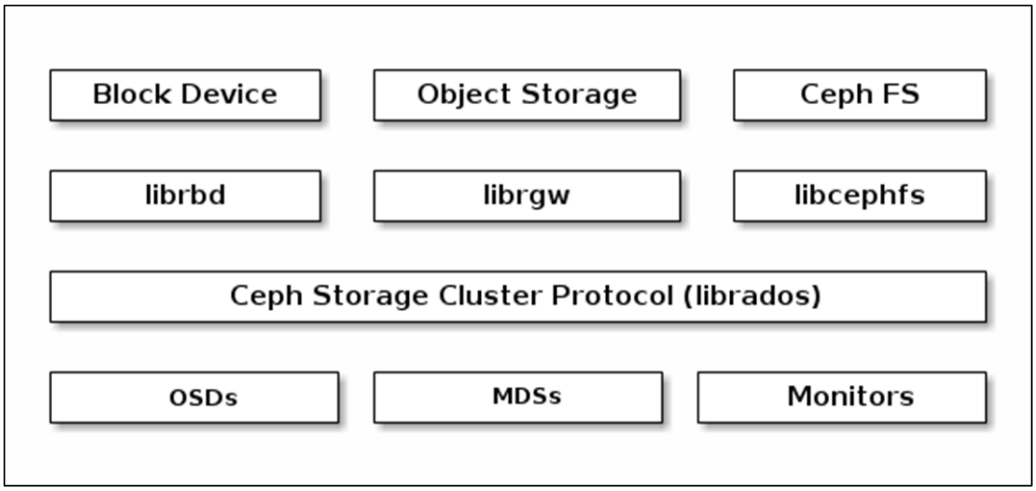
3) MON [Monitor]

Ceph Monitor maintains maps of the cluster state, including the monitor map, the OSD map, the Placement Group(PG) map, and the CRUSH map. Ceph maintains a history(epoch) of each state change in the Ceph Monitors, Ceph OSD Daemons, and PGs.

Ceph Architecture

Ceph uniquely delivers block, object, and file storage in one unified system. We can use the same Ceph cluster to operate them simultaneously

i) Ceph Block Device
ii) Ceph Object Gateway
iii) Ceph Filesystem



Ceph architecture diagram

i) Ceph Block Device

Ceph block devices are thin-provisioned, resizable and store data striped over multiple OSDs in a Ceph cluster. Ceph block devices leverage RADOS capabilities such as snapshotting, replication and consistency. Ceph's RADOS Block Devices(RBD) interact with OSDs using kernel modules or the librbd library.

Ceph's block devices deliver high performance with infinite scalability, to kernel modules, to KVMs such as QEMU, and cloud-based computing systems like OpenStack and CloudStack that rely on libvirt and QEMU to integrate with Ceph block devices.

Ceph Block Device supports caching. Kernel modules can use Linux page caching. While librbd-based applications use RBD Caching.

Ceph Block Device does NOT use Ceph Metadata Server

ii) Ceph Object Gateway

This is an object storage interface built on top of librados[1][2] to provide applications with a RESTful gateway to Ceph Storage Clusters.

Ceph Object Gateway together with Ceph Storage Cluster is called Ceph Object Storage. Ceph Object Storage supports two interfaces.

- S3-compatible - Provides object storage functionality with an interface that is compatible with a large subset of the Amazon S3 RESTful API
- Swift-compatible - Provides object storage functionality with an interface that is compatible with a large subset of the OpenStack Swift API

Ceph Object Storage uses the Ceph Object Gateway daemon(radosgw), which is a FastCGI module for interacting with Ceph Storage Cluster. Since it provides interfaces compatible with S3 and Swift, the Ceph Object Gateway has its own user management. Ceph Object Gateway can store data in the same Ceph Storage Cluster used to store data from Ceph Filesystem clients or Ceph Block Device clients. As the S3 and Swift APIs share a common namespace, we may write data with one API and retrieve it with the other.

Ceph Object Storage does NOT use Ceph Metadata Server

iii) Ceph Filesystem

Ceph Filesystem or Ceph FS is a POSIX compliant filesystem that uses a Ceph Storage Cluster to store its data. It requires at least one Ceph Metadata Server in our Ceph Storage Cluster.

NOTE: http://docs.ceph.com/docs/master/cephfs/best-practices/

We will be using Ceph Block Device as our storage mechanism for Ceph with OpenStack. To use Ceph Block Devices with OpenStack, we must install QEMU, libvirt, and OpenStack first.
Three parts of OpenStack integrate with Ceph's block devices:

- Images
- Volumes
- Guest Disks

Ceph does not support QCOW2 for hosting a virtual machine disk. So if we want to boot virtual machines in Ceph (ephemeral backend or boot from volume), the Glance image format must be RAW.

Ceph block devices, by default uses the rbd pool. We may use any available pool. We recommend creating a pool for Cinder and a pool for Glance.

OS Installation

We will be using Ubuntu 16.04.1 LTS (Xenial Xerus) 64-bit Server Edition for our nodes.
This edition of Ubuntu was released on ***2016-04-21*** and is supported until ***2021-04***.

Download the corresponding ISO from http://releases.ubuntu.com/xenial/.
We have downloaded the ISO file ***ubuntu-16.04.1-server-amd64.iso***.

Prepare a bootable media using the ISO.
If optical media is being used, prepare the bootable media using appropriate application.
But if we are using HDD or SSD, bootable media may be created using following method.
Open a terminal in the directory containing ISO file. Execute the command with following syntax.

SYNTAX:

```
# dd if=ubuntu-16.04.1-server-amd64.iso of=/dev/sdaX
```

Replace **X** with the drive letter of HDD or SSD device

Our Ceph Storage Cluster consists of one Ceph admin node and two Ceph OSD nodes as shown below.

Ceph admin node	⟹	10.3.0.3 [***cinder-ceph.office.brookwin.com***]
Ceph OSD node 1	⟹	10.3.0.4 [***ceph-osd1.office.brookwin.com***]
Ceph OSD node 2	⟹	10.3.0.5 [***ceph-osd2.office.brookwin.com***]

Let us start with installing Ubuntu on each nodes. Boot the nodes with the bootable media we just created.

1) Upon initial booting we will presented a choice to select the Language, with ***English*** highlighted as default.
Make sure ***English*** is selected and press Enter.

2) Next we will see the installation splash screen with Boot menu. We can initiate the installation of OS from here.
Select ***Install Ubuntu Server*** and press Enter.

3) Next will be the Language selection screen with ***English*** highlighted as the default. This language will be used for the installation process as well as the default language for installed system.
Make sure ***English*** is selected and press Enter.

4) Next will be the Location selection screen. Here we can select the location for our server
Choose ***India*** and press Enter.

5) On next screen, the installer will ask us whether we want it to detect our keyboard layout or we want to choose from a list of available options. Let us manually configure the keyboard.

Select ***No*** and press Enter.

6) On the next screen, the installer will list the country of origin of our keyboard.

Select ***English(US)*** and press Enter.

7) This will be followed by a screen that lists countries that follows different keyboard layouts. Our keyboard layout language is English and is followed by US.

Select ***English (US)*** and press Enter.

This will be followed by detection of hardware, after which installer will load installation components from media.

The installer will further detect network hardware, after which we will be guided through the Network Configuration setup.

8) If there is more than one network interface(*which we surely has for the Ceph admin node*), we will be given an option to select the primary network interface.

We may choose an option of our choice and press Enter.

9) If our network has DHCP configured, it will detect and automatically configure the network.

Our network has DHCP server available. So the network will be automatically configured.

10) On the next screen, we will be able to specify our server's hostname. Specify the hostname appropriately for each node.

NOTE:

Ceph admin node	⟹	10.3.0.3 ***[cinder-ceph.office.brookwin.com]***
Ceph OSD node 1	⟹	10.3.0.4 ***[ceph-osd1.office.brookwin.com]***
Ceph OSD node 2	⟹	10.3.0.5 ***[ceph-osd2.office.brookwin.com]***

After specifying the hostname, select ***Continue*** and press Enter.

After the Network is configured, there will be steps to setup Users and Passwords.

11) We will be presented with a screen where we can specify the user's real name.

 Specify our user's real name as **user1**

Select ***Continue*** and press Enter.

12) On next screen, we can specify a username for our user. The username should start with a lower case letter, which can be followed by any combination of numbers and more lower case letters.

 Specify the username as **user1**

Select ***Continue*** and press Enter.

13) On next screen, we can specify a password for our user.

 Set the password as **101010**

Select ***Continue*** and press Enter.

14) This will be followed by a password verification screen.

 Set the password as **101010**

Select ***Continue*** and press Enter.

15) Since we have used a weak password, the installer will prompt us if we should proceed with this password. We are doing this for testing purposes and is fine to do so.

Select ***Yes*** and press Enter.

16) Next, the installer will ask us if we want to encrypt the home directory of our user. We will not be needing this here.

Select ***No*** and press Enter.

After Users and Passwords are setup, the installer will proceed with setting up Time, Date and Timezone.

17) First installer will go for retrieving Time and Date. If there is no internet connection, time will be retrieved from the CMOS in motherboard. But if we are connected to internet, installer will get time from the Network Time Server.

Then based on our present physical location, installer will prompt us to confirm a specific Timezone. This has to be be ***Asia/Calcutta*** for us.

Select ***Yes*** and press Enter. The installer will proceed with setting up the retrieved Time and date for our server.

After this comes the important step ➠ Disk Partitioning.

NOTE:

We will be using XFS filesystem for partitions instead of ext4. Starting with the Jewel release Ceph OSDs require XFS filesystem.

ext4 filesystem has limitations in the size of xattrs it can store, and this causes problems with the way Ceph handles long RADOS object names. Although these issues will generally not surface with Ceph clusters using only short object names (e.g: an RBD workload that does not include long RBD image names), other instances like RGW make extensive use of long object names and can break.

Ceph OSD daemons typically will not start on an ext4 filesystem. Though the official Ceph documentation gives a way to tackle this, the daemons still will not start. This could supposedly be a bug.

Since XFS is more robust and is required for OSD nodes, we will be using XFS in all our nodes.

18) We will be presented with a list of choices for Partitioning.
Select the Partitioning method as ***Manual*** and press Enter.

19) Let us setup a new partition for installing the operating system.
Create a new partition.

➜ Select Create a new partition

➜ Select new partition size as ***20 GB***

➜ Select new partition type as ***Primary***

➜ Select Location for the new partition as ***Beginning***

We will be presented with Partition settings for new partition. Configure the options as following.

➜ Set *Use as* to ***XFS journaling file system***

➜ Set *Mount point* to ***/***

➜ If there is such an option *Format the partition*, set it to ***yes, format it***

➜ Set *Bootable flag* to ***on***

Select ***Done setting up the partition***.

20) Next setup a partition for use as swap space.
Create a new partition.

➜ Select Create a new partition

➜ Select new partition size for swap space. Use the Redhat official documentation for determining swap space.

➜ Select new partition type as ***Primary***

➜ Select Location for the new partition as ***Beginning***

We will be presented with Partition settings for new partition. Configure the options as following.

➜ Set *Use as* to ***swap area***

Select ***Done setting up the partition***

21) For both nodes *ceph-osd1* and *ceph-osd2*, after creating partitions for */* and *swap*, create an XFS partition using the remaining disk space.

Create a new partition.

➜ The *New partition size* will show the remaining disk space available. Leave it without making any changes

➜ Select new partition type as ***Primary***

➜ Select Location for the new partition as ***Beginning***

We will be presented with Partition settings for new partition. Configure the options as following.

➜ Set *Use as* to ***XFS journaling file system***

➜ For *ceph-osd1* node, set *Mount point* to ***/mnt/osd1***

OR

➜ For *ceph-osd2* node, set *Mount point* to ***/mnt/osd2***

➜ If there is such an option *Format the partition*, set it to ***yes, format it***

Select ***Done setting up the partition***.

22) After we are done, take an overview of the partitioning setup.

If everything is done accordingly, select ***Finish partitioning and write changes to disk***.

23) The installer will prompt us one more time if it should write the changes to disk.

Select ***Yes*** and press Enter.

The installer will now proceed with installing the base system. This process may take a while.
After the base system is installed, we will be guided to the Software selection and installation.

24) The first screen will be where the installer asks us whether we want to configure a HTTP proxy. We do not plan to do this.

Leave the field blank. Select ***Continue*** and and press Enter.

25) On next screen, we can select how to manage upgrades on the installed system.

Select ***No automatic updates***.

26) The next screen is for Software Selection. Depending on the purpose for which this server will be used we can select different software groups. We must select the following softwares to install.

- ***Virtual Machine Host***
- ***OpenSSH server***

Select ***Continue*** and press Enter.

Depending on the Software groups we have chosen, this may take some time.

After the installation of software is complete, we will be taken to the Bootloader installation screen.

27) The installer will prompt us whether to install GRUB boot loader to the MBR or not.
Select ***Yes*** and press Enter.

28) On the next screen, installer will allow us to select the device for boot loader installation
Select the appropriate device and press Enter.

Finally the installer will show us the Installation complete screen.

Remove the installation media and press ***Continue***.

Our Ubuntu Server operating system should boot into the login prompt.

Basic OS Setup

Let us configure our installed systems.

1) Setup root user

The root user is disabled in a typical Ubuntu installation. Let us enable the root user.

Perform the following step in all nodes.

Set a password for root user. This will also enable the root user.

```
# passwd root
```

Set the password as **101010**

2) Configure Text Editor

We need a versatile and comfortable text editor to work in a command line environment.

Let us use ***vi*** or it's clone ***vim*** for this purpose.

Perform the following step in all nodes.

Enable line number in ***vi*** instances for root user.

```
# echo 'set number' > /root/.vimrc
```

3) Configure networking

Now let us configure networking in all our nodes.
Our Ceph Storage Cluster consists of one Ceph admin node and two Ceph OSD nodes as shown below.

Ceph admin node ⟹ 10.3.0.3 [***cinder-ceph.office.brookwin.com***]
Ceph OSD node 1 ⟹ 10.3.0.4 [***ceph-osd1.office.brookwin.com***]
Ceph OSD node 2 ⟹ 10.3.0.5 [***ceph-osd2.office.brookwin.com***]

Perform the following changes in Ceph admin node.

In interface configuration file ***/etc/network/interfaces***, make the following changes.

```
#auto enp3s0
#iface enp3s0 inet dhcp

auto enp3s0
iface enp3s0 inet static
    address 10.3.0.3
    netmask 255.0.0.0
    network 10.0.0.0
    broadcast 10.255.255.255
    gateway 10.1.0.2
    dns-nameservers 10.1.0.10
    dns-search office.brookwin.com
```

In host configuration file ***/etc/hosts***, make the following changes.

```
#127.0.1.1          cinder-ceph.office.brookwin.com

10.3.0.3            cinder-ceph.office.brookwin.com        cinder-ceph

10.3.0.4            ceph-osd1.office.brookwin.com          ceph-osd1
10.3.0.5            ceph-osd2.office.brookwin.com          ceph-osd2
```

Reboot the system.

Perform the following changes in 1st Ceph OSD node.

In interface configuration file ***/etc/network/interfaces***, make the following changes.

```
#auto enp2s0
#iface enp2s0 inet dhcp

auto enp2s0
iface enp2s0 inet static
    address 10.3.0.4
    netmask 255.0.0.0
    network 10.0.0.0
    broadcast 10.255.255.255
    gateway 10.1.0.2
    dns-nameservers 10.1.0.10
    dns-search office.brookwin.com
```

In host configuration file ***/etc/hosts***, make the following changes.

```
#127.0.1.1    ceph-osd1.office.brookwin.com

10.3.0.4            ceph-osd1.office.brookwin.com          ceph-osd1

10.3.0.3            cinder-ceph.office.brookwin.com        cinder-ceph
10.3.0.5            ceph-osd2.office.brookwin.com          ceph-osd2
```

Reboot the system.

Perform the following changes in 2nd Ceph OSD node.

In interface configuration file ***/etc/network/interfaces***, make the following changes.

```
#auto enp3s0
#iface enp3so inet dhcp

auto enp3s0
iface enp3s0 inet static
    address 10.3.0.5
    netmask 255.0.0.0
    network 10.0.0.0
    broadcast 10.255.255.255
    gateway 10.1.0.2
    dns-nameservers 10.1.0.10
    dns-search office.brookwin.com
```

In host configuration file ***/etc/hosts***, make the following changes.

```
#127.0.1.1          ceph-osd2.office.brookwin.com

10.3.0.5            ceph-osd2.office.brookwin.com       ceph-osd2

10.3.0.3            cinder-ceph.office.brookwin.com     cinder-ceph
10.3.0.4            ceph-osd1.office.brookwin.com       ceph-osd1
```

Reboot the system.

4) Configure SSH server

All the nodes in our cluster requires to be able to perform passwordless SSH login to each other as root user. To enable this, perform the following changes in all nodes.

In OpenSSH Server configuration file ***/etc/ssh/sshd_config***, make the following changes.

Change the logging level from ***INFO*** to ***VERBOSE***.

```
#LogLevel INFO
LogLevel VERBOSE
```

Allow root login using password.

```
#PermitRootLogin prohibit-password
PermitRootLogin yes
```

Restart ***ssh*** service.

```
# systemctl restart ssh
```

5) Configure SSH keys and passwordless SSH login

Now generate the SSH keys. Execute the following command in all nodes.

```
# ssh-keygen -t rsa
```

This will ask for a passphrase. Leave the passphrase blank and press enter two times.

This will generate an SSH private key ***id_rsa*** and SSH public key ***id_rsa.pub*** under directory ***/root/.ssh/***. The keys will follow PEM standard and use RSA public key algorithm using ***2048-bit keysize***. It will be signed using the hash algorithm SHA-256. This will also be accompanied by the key fingerprint and ***randomart image*** being displayed in the standard output.

Finally enable passwordless SSH login on each nodes separately. The below given ssh-copy-id commands does the following in each nodes.

ssh-copy-id will display the ECDSA key fingerprint of the remote host, followed by a ***yes*** or ***no*** prompt; Type ***yes*** and press Enter. Further the root password of remote host will be requested. Type the password and press Enter.

Upon entering the password, ***ssh-copy-id*** copies the identity file which is the local host's public key ***/root/.ssh/id_rsa.pub*** to the remote-host's authorized keys file ***/root/.ssh/authorized_keys***. ssh-copy-id also assigns proper permission to the remote host's home and ***authorized_keys*** file.

Execute the following commands in Ceph admin node.

```
root@cinder-ceph:~# ssh-copy-id -i ~/.ssh/id_rsa.pub root@10.3.0.4
```

```
root@cinder-ceph:~# ssh-copy-id -i ~/.ssh/id_rsa.pub root@10.3.0.5
```

This will enable passwordless root SSH login ***from 10.3.0.3*** to nodes ***10.3.0.4 and 10.3.0.5***.

Execute the following commands in 1st Ceph OSD node.

```
root@ceph-osd1:~# ssh-copy-id -i ~/.ssh/id_rsa.pub root@10.3.0.3
```

```
root@ceph-osd1:~# ssh-copy-id -i ~/.ssh/id_rsa.pub root@10.3.0.5
```

This will enable passwordless root SSH login ***from 10.3.0.4*** to nodes ***10.3.0.3 and 10.3.0.5***.

Execute the following commands in 2nd Ceph OSD node.

```
root@ceph-osd2:~# ssh-copy-id -i ~/.ssh/id_rsa.pub root@10.3.0.3
```

```
root@ceph-osd2:~# ssh-copy-id -i ~/.ssh/id_rsa.pub root@10.3.0.4
```

This will enable passwordless root SSH login ***from 10.3.0.5*** to nodes ***10.3.0.3 and 10.3.0.4***.

6) Configure iptables

Setting up a good firewall is an essential step to secure any modern operating system. Ubuntu ships with iptables as the standard firewall. But in the default setup, the rules that we add to iptables are ephemeral. This means when we restart the server, the iptables rules will be gone. This maybe a feature for some as it gives them an avenue to get back in if they have accidentally locked themselves out of the server. However for most practical applications this is not the desired behavior.

There are a few ways to make the iptables rules persistent across boot. But the following must be noted.

For security, the iptables configuration should be applied at an early stage of the bootstrap process: preferably before any network interfaces are brought up, and certainly before any network services are started or routing is enabled. If this is not done then there will be a window of vulnerability during which the machine is remotely accessible but not firewalled.

So the best way to make iptables rules persistent across boot is with the ***iptables-persistent*** package.

Perform the following changes in all nodes.

Install ***iptables-persistent*** package.

```
# apt install iptables-persistent
```

This will also install the ***netfilter-persistent*** package.

During installation, there will be prompts to save current IPv4 and IPv6 rules to ***/etc/iptables/rules.v4*** and ***/etc/iptables/rules.v6*** respectively. Select ***No*** for both. This is because libvirtd and associated services always activate a certain set of iptables rules when they detect a NAT network(*For us, we are in one*). So restoring the iptables from the *rules* files will create duplicate iptables entries. This is a scenario we do not want. To tackle this we will manually create the file ***/etc/iptables/rules.v4*** and also manually add rules whenever required.

Create file ***/etc/iptables/rules.v4*** with following content.

```
*filter
:INPUT ACCEPT
:FORWARD ACCEPT
:OUTPUT ACCEPT
COMMIT
```

7) Install NTP package

While time passes by, computers internal clock tend to drift which can lead to inconsistent time. This is of concern, especially on servers and clusters where we want to cross check clients logs across nodes or we want to replicate server resources or databases.

NTP or Network Time Protocol comes to the rescue here. Servers can be synced using an NTP daemon that slowly shift the server clock to match a reference time that servers around the world agree upon.

Perform the following step in all nodes.

Install NTP package.

```
# apt install ntp
```

8) Update Package manager metadata cache

Let us update the Package manager metadata cache.

Execute the following command in all nodes.

```
# apt update
```

This will download the package index files from the sources specified in ***/etc/apt/sources.list*** file, along with sources specified in the ***.list*** files in directory ***/etc/apt/sources.list.d/***, and update the metadata cache of Package Manager.

9) Installed Services

cinder-ceph

systemd style

[Active] /lib/systemd/system/netfilter-persistent.service

Sys-V style

[Active] /etc/init.d/ntp

ceph-osd1 & ceph-osd2

systemd style

[Active] /lib/systemd/system/netfilter-persistent.service

Sys-V style

[Active] /etc/init.d/ntp

Installing Ceph Storage Cluster

Let us create a Ceph Storage Cluster with one Ceph Monitor and two Ceph OSD Daemons.

Once the cluster reaches an ***active + clean*** state, if needed we MAY expand it by adding more Ceph OSD daemon(s) OR Metadata Server(s) OR two more Ceph Monitors.

1) Add Ceph repositories
2) Install and Configure Ceph Deploy
3) Create Cluster
4) Add OSDs to Cluster
5) Check for errors
6) Add Monitors to Cluster
7) Check for errors
8) Create Block device

1) Add Ceph repositories

Let us add Ceph repositories to the Ceph admin node.

Perform the following operations in Ceph admin node.

Add the release key for Ceph packages.

```
root@cinder-ceph:~# wget -q -O- 'https://download.ceph.com/keys/release.asc' | apt-key add -
```

Add the Ceph packages to our repository. Execute the following command.

```
root@cinder-ceph:~# echo deb https://download.ceph.com/debian-jewel/ $(lsb_release -sc) main > /etc/apt/sources.list.d/ceph.list
```

Replace ***{ceph-stable-release}*** with a stable Ceph release (e.g., *hammer*, *jewel*, etc.)
We will be using Ceph Jewel release in our setup

SYNTAX:

```
# echo deb https://download.ceph.com/debian-{ceph-stable-release}/ $(lsb_release -sc) main > /etc/apt/sources.list.d/ceph.list
```

Ceph repositories have been added to ***ceph.list*** in the ***/etc/apt/sources.list.d/*** directory.

Let us update the Package manager metadata cache. Execute the following command.

```
root@cinder-ceph:~# apt update
```

This will download the package index files from the sources specified in ***/etc/apt/sources.list*** file, along with sources specified in the ***.list*** files in directory ***/etc/apt/sources.list.d/***, and update the metadata cache of Package Manager.

2) Install and Configure Ceph Deploy

ceph-deploy tool is a way to deploy Ceph relying only upon SSH access to the servers, sudo, and Python. It runs on the workstation, and does not require servers, databases, or any other tools.

If we set up and tear down Ceph clusters a lot, and want minimal extra bureaucracy, ceph-deploy is an ideal tool. With ceph-deploy, we can develop scripts to install Ceph packages on remote hosts, create a cluster, add monitors, gather or forget keys, add OSDs and metadata servers, configure admin hosts, and tear down the clusters.

ceph-deploy tool is not a generic deployment system. It was designed exclusively for Ceph users who want to get Ceph up and running quickly with sensible initial configuration settings without the overhead of installing additional third party tools.

NOTE:

Users who want fine-control over security settings, partitions or directory locations should use a tool such as Juju, Puppet, Chef or Crowbar

Do not call ***ceph-deploy*** with sudo or run it as root if you are logged in as a different user, because ceph-deploy will not issue sudo commands needed on the remote host

We will be using ***ceph-deploy*** in this setup.

Let us install ceph-deploy in the Ceph admin node. Execute the following command in Ceph admin node.

```
root@cinder-ceph:~# apt install ceph-deploy
```

ceph-deploy relies on python for installing Ceph. Ceph admin node already has python installed as a dependency for ceph-deploy. But the Ceph OSDs still needs python to be installed.

So let us install python in the Ceph OSDs. Execute the following command on both Ceph OSDs.

```
# apt install python
```

The ceph-deploy utility must login to a Ceph node as a user that has password less sudo privileges, because it needs to install software and configuration files without prompting for passwords. It is recommend to create a specific user for ceph-deploy on all Ceph nodes in the cluster.

- ✓ Do not use ***ceph*** as the user name. Starting with the Infernalis release the ***ceph*** user name is reserved for Ceph daemons
- ✓ If the ***ceph*** user already exists on Ceph nodes, removing the user must be done before attempting an upgrade

Let us create a user named ***cephdeploy*** in the admin node for this purpose.

Perform the following operations in Ceph admin node.

Create a user ***cephdeploy*** with home directory being ***/home/cephdeploy***. If this directory does not exist, it will be created, and contents of the skeleton directory will be copied to the home directory.

```
root@cinder-ceph:~#  useradd -d /home/cephdeploy -m cephdeploy
```

Set a password for user ***cephdeploy***.

Enter password as **101010**

```
root@cinder-ceph:~#  passwd cephdeploy
```

Create a file named ***cephdeploy*** in the directory ***/etc/sudoers.d/*** with following content.

```
cephdeploy ALL = (root) NOPASSWD:ALL
```

This means that, the cephdeploy user on any host may run any command as root without a password.
The first ALL refers to hosts, while the last ALL refers to allowed commands
Execute the following command.

```
root@cinder-ceph:~#  echo "cephdeploy ALL = (root) NOPASSWD:ALL" >
/etc/sudoers.d/cephdeploy
```

Set permissions for ***/etc/sudoers.d/cephdeploy*** file to ***0440***.

```
root@cinder-ceph:~#  chmod 0440 /etc/sudoers.d/cephdeploy
```

Now let us set the netfilter/iptables firewall rules for Ceph.

Ceph Monitors communicate using port ***6789*** by default.
Open the port ***6789*** for Ceph monitor nodes. Execute below command in all nodes.

```
#  iptables -A INPUT -p tcp --dport 6789 -j ACCEPT
```

```
#  sed -i '$i-A INPUT -p tcp --dport 6789 -j ACCEPT' /etc/iptables/rules.v4
```

Ceph OSDs communicate within the port range ***6800 to 7300*** by default.
Open the port range ***6800 to 7300*** for both Ceph OSDs. Execute below command in both Ceph OSDs.

```
#  iptables -A INPUT -p tcp --match multiport --dports 6800:7300 -j ACCEPT
```

```
#  sed -i '$i-A INPUT -p tcp --match multiport --dports 6800:7300 -j ACCEPT'
/etc/iptables/rules.v4
```

3) Create Cluster

Let us create our first Ceph Cluster.

Perform the following operations on Ceph admin node.

We will be deploying the cluster using ceph-deploy.
ceph-deploy requires a directory for holding the configuration files that it deploys along the cluster.
Create a directory named ***cluster1*** for maintaining the configuration files and keys that ceph-deploy generates for the cluster.

```
root@cinder-ceph:~# mkdir cluster1
```

Change to the new directory ***cluster1*** in terminal.

```
root@cinder-ceph:~# cd cluster1
```

Create the Ceph cluster.

```
root@cinder-ceph:~# ceph-deploy new cinder-ceph
```

In directory ***cluster1***, we should see the following contents:

- ceph.conf ⟹ *Configuration file that will be pushed to all nodes while deploying cluster*
- ceph.mon.keyring ⟹ *Keyring file used by monitoring nodes for communicating with each other*
- ceph-deploy-ceph.log ⟹ *Logfile used by ceph-deploy*

Change the default number of replicas in the Ceph configuration file from 3 to 2 so that the Ceph cluster can achieve an ***active + clean*** state with just two Ceph OSDs.

In configuration file ***ceph.conf***, add the following line under ***[global]*** section.

```
osd pool default size = 2
```

The default RBD image features have been updated to enable the following[source]: *exclusive lock*, *object map*, *fast-diff*, and *deep-flatten*. These features are not currently supported by the RBD kernel driver nor older RBD clients. Update the default features to pre-Jewel setting.

In configuration file ***ceph.conf***, create a ***[client]*** section and add the following line.

```
rbd default features = 1
```

NOTE:

A Ceph Storage Cluster requires at least one Ceph Monitor to run.
But for high availability, Ceph Storage Clusters typically run multiple Ceph Monitors so that the failure of a single monitor will not bring down the whole Cluster down. Ceph uses the Paxos algorithm, which requires a majority of monitors (i.e., *1, 2:3, 3:4, 3:5, 4:6*, etc.) to form a quorum.
In this scenario, we are building a High availability Ceph Storage Cluster with 3 Monitors. We are first aiming to setup a Ceph cluster with 1 Monitor and 2 OSDs. After we have achieved an ***active + clean*** state, we will be expanding the cluster by adding 2 more Monitors.
However, in case we wanted to add all the monitors at once, the configuration file ***ceph.conf*** would have contained the following entries.

```
mon_initial_members = cinder-ceph,ceph-osd1,ceph-osd2
mon_host = 10.3.0.3,10.3.0.4,10.3.0.5
public_network = 10.3.0.0/24
```

Install Ceph using below command.

```
root@cinder-ceph:~# ceph-deploy install cinder-ceph ceph-osd1 ceph-osd2
```

This will install Ceph on each node.
In directory ***cluster1***, we should see Ceph's PGP key ***release.asc*** key. This key will be used to verify the packages downloaded by Ceph.

Add the initial monitor(s) and gather the keys.

```
root@cinder-ceph:~# ceph-deploy mon create-initial
```

Once we complete this process, the directory ***cluster1*** should have the following keyrings

- ceph.bootstrap-mds.keyring — ***To generate cephx keyrings for MDS instances***
- ceph.bootstrap-osd.keyring — ***To generate cephx keyrings for OSD instances***
- ceph.bootstrap-rgw.keyring — ***To generate cephx keyrings for RGW instances***
- ceph.client.admin.keyring — ***To administer Ceph cluster by ceph client commands***

4) Add OSDs to Cluster

Let us add two OSDs to our Ceph Cluster.
These are gonna be *ceph-osd1.office.brookwin.com* and *ceph-osd2.office.brookwin.com*.

The Ceph OSDs need a location to store the cluster data.

For *ceph-osd1.office.brookwin.com*, we will use rest of the disk space mounted on ***/mnt/osd1/***.
Ceph daemons will need access to this location. So let us set the owner of ***/mnt/osd1/*** as ***ceph***.

```
root@ceph-osd1:~# chown -R ceph /mnt/osd1
```

And for *ceph-osd2.office.brookwin.com*, we will use rest of the disk space mounted on ***/mnt/osd2/***.
Give ceph daemons access to this location. Let us set the owner of ***/mnt/osd2/*** as ***ceph***.

```
root@ceph-osd2:~# chown -R ceph /mnt/osd2
```

Then, from our Ceph admin node, use ***ceph-deploy*** to prepare the OSDs.

```
root@cinder-ceph:~# ceph-deploy osd prepare ceph-osd1:/mnt/osd1 ceph-osd2:/mnt/osd2
```

Finally, activate the OSDs also from our Ceph admin node.

```
root@cinder-ceph:~# ceph-deploy osd activate ceph-osd1:/mnt/osd1 ceph-osd2:/mnt/osd2
```

Copy the configuration file ***ceph.conf*** and admin keyring ***ceph.client.admin.keyring*** to all nodes so that we can use the ceph commands without having to specify the monitor address and admin keyring each time.

```
root@cinder-ceph:~# ceph-deploy admin cinder-ceph ceph-osd1 ceph-osd2
```

When ***ceph-deploy*** is talking to the local admin node, it must be reachable by its hostname

Ensure that read permission for the admin keyring ***ceph.client.admin.keyring*** in enabled all nodes.
Execute the following command in all nodes.

```
# chmod +r /etc/ceph/ceph.client.admin.keyring
```

Finally let us check the details of the Ceph cluster.

Perform the following operation in any of the Ceph nodes.

Check the status of Ceph Storage Cluster.

```
# ceph status
```

The cluster should return an ***active + clean*** state when it has finished peering.

5) Check for errors

Finally, let us check if there are any errors in our deployment of Ceph.
Perform the following operations in all nodes.

Copy the directory ***/var/log/ceph*** as a backup renamed with current date and time, delete any existing log files from directory ***/var/log/ceph***, and reboot the server.

```
# cp -avr /var/log/ceph /var/log/ceph_bak_$(date +"%Y%m%d-%H%M%S") && find /var/log/ceph -type f | xargs rm -f && reboot
```

After the server comes up, check for any critical messages or errors or warnings.

```
# grep crit -irl /var/log/ceph
```

```
# grep err -irl /var/log/ceph
```

```
# grep warn -irl /var/log/ceph
```

6) Add Monitors to Cluster

Our Ceph cluster currently has a single Ceph monitor, which is the Ceph admin node itself.

Now let us add 2 additional Monitors to our Ceph Cluster.
This will be the nodes *ceph-osd1.office.brookwin.com* and *ceph-osd2.office.brookwin.com*.

Perform the following operations in Ceph admin node.

Change to the directory ***cluster1***.

```
root@cinder-ceph:~# cd cluster1
```

In configuration file ***ceph.conf***, make the following changes under ***[global]*** section.

Add *ceph-osd1* and *ceph-osd2* as initial monitors.
Modify value of ***mon_initial_members*** and ***mon_host*** as follows

```
mon_initial_members = cinder-ceph,ceph-osd1,ceph-osd2
mon_host = 10.3.0.3,10.3.0.4,10.3.0.5
```

Define the public network under which Ceph Monitors belong.

```
public_network = 10.3.0.0/8
```

Finally, add the Ceph monitors to quorum.

```
root@cinder-ceph:~# ceph-deploy --overwrite-conf mon create ceph-osd1 ceph-osd2
```

NOTE:

We may get the following errors while adding Ceph monitors.

```
[ceph-osd1][WARNIN] monitor ceph-osd1 does not exist in monmap
```

```
[ceph-osd2][WARNIN] monitor ceph-osd2 does not exist in monmap
```

This is supposedly a bug related to ceph-deploy. If we receive this error, it is recommended to run the same command immediately once again. Otherwise this can cause errors for Placement Groups maps.

```
root@cinder-ceph:~# ceph-deploy --overwrite-conf mon create ceph-osd1 ceph-osd2
```

Once we have added the Ceph Monitors, Ceph will begin synchronizing the monitors and form a quorum. We can check the quorum status by executing the following command in any of the nodes.

```
# ceph quorum_status -f json-pretty
```

7) Check for errors

Perform the following operations in all nodes.

Copy the directory ***/var/log/ceph*** as a backup renamed with current date and time, delete any existing log files from directory ***/var/log/ceph***, and reboot the server.

```
# cp -avr /var/log/ceph /var/log/ceph_bak_$(date +"%Y%m%d-%H%M%S") && find /var/log/ceph -type f | xargs rm -f && reboot
```

After the server comes up, check for any critical messages or errors or warnings.

```
# grep crit -irl /var/log/ceph
```

```
# grep err -irl /var/log/ceph
```

```
# grep warn -irl /var/log/ceph
```

8) Create a Block device

Now we have created a fully working Ceph cluster, let us configure a Block device on the cluster.

Ensure that the Ceph Storage Cluster is in ***active + clean*** state before working with Ceph Block Device.

```
# ceph status
```

The cluster should return an ***active + clean*** state on the above command.

Perform the following operation in any of the Ceph OSDs.

Create a block device image of size ***1024 MB***. Let us name the block device image as ***block1***.

```
# rbd create block1 --size 1024
```

Map the block device image ***block1*** to a block device.

```
# rbd map block1 --pool rbd --name client.admin
```

EXPECTED OUTPUT:

```
/dev/rbd0
```

Format the block device with XFS filesystem.

```
# mkfs.xfs /dev/rbd/rbd/block1
```

EXPECTED OUTPUT:

```
meta-data=/dev/rbd/rbd/block1    isize=512    agcount=9, agsize=31744 blks
         =                       sectsz=512   attr=2, projid32bit=1
         =                       crc=1        finobt=1, sparse=0
data     =                       bsize=4096   blocks=262144, imaxpct=25
         =                       sunit=1024   swidth=1024 blks
naming   =version 2              bsize=4096   ascii-ci=0 ftype=1
log      =internal log           bsize=4096   blocks=2560, version=2
         =                       sectsz=512   sunit=8 blks, lazy-count=1
realtime =none                   extsz=4096   blocks=0, rtextents=0
```

Create a directory ***/mnt/block1/*** for mounting the block device.

```
# mkdir /mnt/block1
```

Mount the block device ***/dev/rbd/rbd/block1/*** at ***/mnt/block1***/

```
# mount /dev/rbd/rbd/block1 /mnt/block1
```

Finally we have our Ceph block device.

Installing Calamari

Calamari is a management and monitoring system for Ceph storage cluster, exposing a high level REST API. It provides a Dashboard User Interface that makes Ceph cluster monitoring amazingly simple and handy.

Calamari was initially a part of Inktank's Ceph Enterprise product offering. But on 30 April 2014 Red Hat announced that it would acquire Inktank Storage. After the acquisition, Red Hat open sourced Calamari. Hence it is an Open Sourced and matured enterprise level software which is being actively developed.

Calamari consists of two components: Calamari Server & Calamari Clients
Both components are maintained as separate Github repositories.

1) Calamari Server (Backend)

Calamari Server forms the backend. It instantiates a new REST API for integration with other systems, whereas earlier versions of Calamari were based on the Ceph REST API. The new Calamari REST API is designed to be much more comprehensive and should be the basis for new development efforts that wish to interact with a Ceph cluster. Source Code is available at http://github.com/ceph/calamari/.

Calamari Server is written in Python using the following components.

- Django (http://djangoproject.com)
- Django REST framework (http://django-rest-framework.org)
- gevent (http://gevent.org)
- Graphite (http://graphiteapp.org)
- SaltStack (http://saltstack.com)
- zerorpc (http://zerorpc.io)

Calamari Server includes the following components.

- Apache HTTP Server (http://httpd.apache.org)
- Carbon cache (http://github.com/graphite-project/carbon)
- cthulhu (http://github.com/ceph/calamari/tree/master/cthulhu)
- PostgreSQL Server (http://postgresql.org)
- salt-master (http://saltstack.com)
- supervisord (http://supervisord.org)

2) Calamari Clients (Frontend)

Calamari Clients forms the frontend. This is a web browser UI implemented primarily in Javascript that uses the Calamari REST API. Source Code is available at http://github.com/ceph/calamari-clients/.

Calamari Client side includes the following components.

- salt-minion (http://saltstack.com)
- diamond (http://github.com/ceph/Diamond/tree/calamari)

As of now official precompiled packages for Calamari are not available. We should build the Calamari Server and Client packages from source. Calamari includes Vagrant build environment for following OSes at present***(2017 March).***

- CentOS 6
- CentOS 7
- RHEL 6 [Santiago] and older
- RHEL 7 [Maipo]
- Ubuntu 12.04 LTS [Precise Pangolin]
- Ubuntu 14.04 LTS [Trusty Tahr]
- Debian 7 [Wheezy]

The Vagrant environment is primarily provided because we do not want to turn our production server into a development environment. Instead one could install the development tools in required Vagrant instance and build the Calamari packages inside them. Once the build process is complete both the Vagrant instances and boxes can be removed.

We are using Ubuntu 16.04.1 LTS [Xenial Xerus]. We could try building the Calamari Debian Packages using Ubuntu 14.04 LTS Vagrant build environment. But the executable ***calamari-ctl*** will fail with following error.

```
ImportError: No module named datetime
```

This is because the Calamari uses Python included in the python virtual environment provided by Calamari debian package. This virtual environment is created based on Ubuntu 14.04 which uses Python 2.7.6 that relies on an external datetime library but that which does not get added to the virtual environment.

Contrast to this Ubuntu 16.04***(16.04.1)*** includes Python 2.7.12 which uses datetime as a built-in module. This means it does not rely on an external library. When we run Calamari based on Ubuntu 14.04, the included python 2.7.6 will look for external datetime library, which of course it will not find. Hence the error.

The solution is to manually build Calamari packages with Python 2.7.12 in Ubuntu 16.04.

And it is a better option to do so considering the following bandwidth and disk usage that Vagrant consumes.

- Vagrant requires a provider to work. By default it has support for providers VirtualBox, Hyper-V, and Docker. If we are planning to use Virtualbox this will need the following in a Ubuntu 16.04.1 OS ⟹ 239 packages(http://pastebin.com/raw/5MkXd9KL) having a download size of 167MB, taking up 661MB of disk space
- 2 Ubuntu 14.04 and 1 Ubuntu 12.04 LTS vagrant boxes amounting to 1.2G in download
- 3 Virtualbox VMs taking 10G diskspace after build process

Following instructions will guide on how to build required packages for Calamari and how to install and configure a working Calamari setup.

1) Building Packages

We may build Calamari packages in any of the nodes. But a separate development server with the same OS and architecture(***Ubuntu 16.04 Server x86_64***) is recommended.

1a) Build Calamari Server package

Building Calamari Server package involves the following steps.

1a1) Install development packages

Let us install the following dependencies required to build Calamari Server packages.

Install Debian package development tools.

```
# apt install dpkg-dev
```

Install debhelper package. It is a collection of programs that can be used in a ***debian/rules*** file to automate common tasks related to building debian packages.

```
# apt install debhelper
```

Install Python development package, Alternative Python package installer, and Python virtual environment creator.

```
# apt install python-dev python-pip python-virtualenv
```

Upgrade the Alternative Python package installer. This will upgrade pip from version ***8.1.1*** to version ***9.0.1***, which is the latest at present(***2017 March***).

```
# pip install --upgrade pip
```

Install development package for Cairo 2D graphics library.

```
# apt install libcairo2-dev
```

Install development package for PostgreSQL C client library.

```
# apt install libpq-dev
```

1a2) Clone Git repository

Now we need a location for cloning Calamari Server Git repository.

In directory ***/tmp/***, create another directory ***calamarirepo***.

```
# mkdir /tmp/calamarirepo
```

Change to the directory ***/tmp/calamarirepo/*** in terminal.

```
# cd /tmp/calamarirepo
```

Clone the calamari Git repository.

```
# git clone https://github.com/ceph/calamari.git
```

This will clone the git repository into directory ***calamari***.

1a3) Build process

Change to the directory ***calamari*** in terminal.

```
# cd calamari
```

The build process is configured to also generate the Calamari source package along with RPM/DEB package.

The source format is set to ***3.0 (quilt)*** in file ***debian/source/format***. If we proceed with this source format, we will encounter the following error.

```
dpkg-source: error: can't build with source format '3.0 (quilt)': no upstream tarball found at ../calamari_1.0.0.orig.tar.{bz2,gz,lzma,xz}
```

This is because the ***3.0 (quilt)*** source format is designed to use upstream tarball. The ***Makefile*** has upstream tarball name set to ***{name}_{version}.orig.tar.gz***. But we have no upstream package with that name. Hence the error.

To tackle this, let us configure the build process to follow ***3.0 (native)*** format.
The file ***debian/source/format*** contains the line ***3.0 (quilt)***.

Replace it with the following line.

```
3.0 (native)
```

Now, if we proceed with this we will get the following error.

```
no such option: --no-install
```

Because the ***Makefile*** contains following line.

```
./bin/python ./bin/pip install --no-install carbon; \
```

The option ***--no-install*** was removed in pip version 7.0.0 (http://pip.pypa.io/en/stable/news/). An equivalent option for ***install --no-install*** was suggested as ***unpack*** in issue 906 but was never implemented.

To bypass this, let us install Carbon cache in the system.

```
# pip install carbon
```

As the system has now Carbon cache installed and thus requirement satisfied, this section of code will be skipped during the make process.

Finally, build the debian packages.

```
# dpkg-buildpackage
```

The build process will output following files in the directory ***/tmp/calamarirepo/***.

- calamari_1.0.0-1_amd64.changes **[Debian changes file]**
- calamari_1.0.0-1.dsc **[Debian Source Control file]**
- calamari_1.0.0-1.tar.xz **[Archived Upstream source tarball]**
- calamari-server_1.0.0-1_amd64.deb **[Debian package]**

But we only need the following file for installing Calamari Server.

- calamari-server_1.0.0-1_amd64.deb

1b) Build Calamari Clients package

Building Calamari Clients package involves the following steps.

1b1) Install development packages
1b2) Clone Git repository
1b3) Build process

1b1) Install development packages

Let us install the following dependencies required to build Calamari Clients package.

Install Ruby, and Compass Stylesheet Authoring Framework.

```
# apt install ruby ruby-compass
```

Install Node.js Package manager.

```
# apt install npm
```

The Calamari Clients source has official support only upto Ubuntu version 14.04. In this version of OS the Node.js binary was named ***node***. But in Ubuntu 16.04***(16.04.1)*** this binary has been renamed to ***nodejs***. So the build process will look for executable ***node*** and fail with the following error.

```
sh: 1: node: not found
```

Let us create an appropriate symlink to fix this issue.

```
# ln -s /usr/bin/nodejs /usr/bin/node
```

Using npm, install Bower which is the Package management solution for frontend components.

```
# npm install -g bower
```

Using npm, install grunt which is The Javascript Task Runner.

```
# npm install -g grunt
```

1b2) Clone Git repository

Change to the directory ***/tmp/calamarirepo/*** in terminal.

```
# cd /tmp/calamarirepo
```

Clone the calamari-clients Git repository.

```
# git clone https://github.com/ceph/calamari-clients.git
```

This will clone the git repository into the directory ***calamari-clients***.

1b3) Build process

Change to the directory ***calamari-clients*** in terminal.

```
# cd calamari-clients
```

If we continue with the build process, it will fail with errors related to npm modules ***grunt-contrib-compass*** and ***grunt-contrib-imagemin***. This is supposedly a problem with ***grunt-contrib-compass*** module.

Execute the following command in terminal. This will list the following files with corresponding entries.

```
# grep -ir "grunt-contrib-imagemin" . | grep '~' | grep -v node_modules
```

EXPECTED OUTPUT:

```
./admin/package.json:    "grunt-contrib-imagemin": "~0.1.4",
./login/package.json:    "grunt-contrib-imagemin": "~0.1.3",
./manage/package.json:    "grunt-contrib-imagemin": "~0.3.0",
```

The file ***package.json*** is a JavaScript Object Notation file that holds the metadata about NPM modules. It has a section ***devDependencies***, which lists the packages that are needed for development and testing. Here the package.json for admin, login and manage clients lists the required versions as 0.1.4, 0.1.3 and 0.3.0 respectively with minor version changes only.

Let us check the current version of ***grunt-contrib-imagemin*** npm module.

```
# npm search grunt-contrib-imagemin | grep ^grunt-contrib-imagemin | awk '{print $6}'
```

EXPECTED OUTPUT:

```
1.0.1
```

The current version of ***grunt-contrib-imagemin*** is version ***1.0.1***, which is a major version change. As per the semver semantics this version will not be accepted. So let us change the major version required in all ***package.json*** file to ***1.0.1***. The resulting line in all ***package.json*** files should look as following.

```
"grunt-contrib-imagemin": "~1.0.1",
```

Finally, build the Calamari clients.

```
# make build-real
```

If all goes well, the client files will be created under following directories.

- ➜ ***admin/dist/***
- ➜ ***dashboard/dist/***
- ➜ ***login/dist/***
- ➜ ***manage/dist/***

Create another directory named ***calamari-clients*** with subdirectories ***admin***, ***dashboard***, ***login*** and ***manage***.

```
# mkdir -p calamari-clients/admin calamari-clients/dashboard calamari-clients/login calamari-clients/manage
```

Copy the contents as following

- ⟹ of ***admin/dist/*** directory to ***calamari-clients/admin/*** directory
- ⟹ of ***dashboard/dist/*** directory to ***calamari-clients/dashboard/*** directory
- ⟹ of'***login/dist/*** directory to ***calamari-clients/login/*** directory
- ⟹ of ***manage/dist/*** directory to ***calamari-clients/manage/*** directory

```
# cp -prv admin/dist/* calamari-clients/admin
```

```
# cp -prv dashboard/dist/* calamari-clients/dashboard
```

```
# cp -prv login/dist/* calamari-clients/login
```

```
# cp -prv manage/dist/* calamari-clients/manage
```

Change to the directory ***calamari-clients*** in terminal.

```
# cd calamari-clients
```

Create a gzipped tarball by archiving the directories ***admin***, ***dashboard***, ***login***, ***manage***.

```
# tar -czpvf ../../calamari-clients.tar.gz admin dashboard login manage
```

This will create the following file in directory ***/tmp/calamarirepo/***.

- ➠ calamari-clients.tar.gz

This is our Calamari Clients package.

1c) Build Diamond package

Diamond helps provide IOPS and other data to the Calamari Interface.
Building Calamari Server package involves the following steps.

1c1) Install development packages
1c2) Clone git repository
1c3) Build process

1c1) Install development packages

Let us install the following dependencies required to build Diamond packages.

Install Common build system for Debian packages.

```
# apt install cdbs
```

Install Python Mocking and Testing Library, and Python config file reader and writer.

```
# apt install python-mock python-configobj
```

The build process also requires ***python-support*** package. But if we try to install python-support via apt, we will receive the following error.

```
Package 'python-support' has no installation candidate
```

This is because package python-support has been removed from Ubuntu 16.04. The functionality has been replaced by package ***dh-python***(http://bugs.launchpad.net/ubuntu/+source/python-support/+bug/1577172).

Strangely, the package python-support is present in repository cache. If we look for amd64 binary package of python-support in https://launchpad.net/ubuntu/xenial/amd64/python-support/1.0.15, it shows the Status as ***Deleted***. Luckily for us, the source package is still archived and maintained.

Let us download and compile the ***python-support*** debian package from source package.

Change to the directory ***/tmp/calamarirepo/*** in terminal.

```
# cd /tmp/calamarirepo
```

Download ***python-support*** source package from http://launchpad.net.

```
# wget http://launchpad.net/ubuntu/+archive/primary/+files/python-support_1.0.15.tar.gz
```

Extract the gzipped tarball.

```
# tar -zxvf python-support_1.0.15.tar.gz
```

Change to the directory ***python-support-1.0.15*** in terminal.

```
# cd python-support-1.0.15
```

Build the debian package.

```
# dpkg-buildpackage
```

The build process will output following files in the directory ***/tmp/calamarirepo/***.

- ➟ python-support_1.0.15_all.deb **[Debian package]**
- ➟ python-support_1.0.15_amd64.changes **[Debian changes file]**
- ➟ python-support_1.0.15.dsc **[Debian Source Control file]**
- ➟ python-support_1.0.15.tar.gz **[Gzipped Upstream source tarball]**
- ➟ python-support_1.0.15.tar.xz **[Archived Upstream source tarball]**

But we only need the following file for installing python-support.

- ➟ python-support_1.0.15_all.deb

Change to directory ***/tmp/calamarirepo/*** in terminal.

```
# cd /tmp/calamarirepo
```

Install python-support package.

```
# dpkg -i python-support_1.0.15_all.deb
```

1c2) Clone Git repository

Change to directory ***/tmp/calamarirepo/*** in terminal.

```
# cd /tmp/calamarirepo
```

Clone the ***calamari*** branch of diamond.

```
# git clone https://github.com/ceph/Diamond.git --branch=calamari
```

This will clone the git repository into directory ***Diamond***.

1c3) Build process

Change to the directory 'Diamond' in terminal.

```
# cd Diamond
```

Build the debian package.

```
# dpkg-buildpackage
```

The build process will output following files in the directory /tmp/calamarirepo/.

- ➠ diamond_3.1.0_all.deb **[Debian package]**
- ➠ diamond_3.1.0_amd64.changes **[Debian changes file]**
- ➠ diamond_3.1.0.dsc **[Debian Source Control file]**
- ➠ diamond_3.1.0.tar.gz **[Gzipped Upstream source tarball]**

But we only need the following file for installing Diamond.

- ➠ diamond_3.1.0_all.deb

1d) Build SaltStack packages

Building SaltStack packages involves the following steps.

1d1) Install development packages
1d2) Clone Git repository
1d3) Build process

1d1) Install development packages

Let us install the following dependencies required to build SaltStack packages.

Install Python bindings for cryptographic algorithms and protocols, Python bindings for Jinja2, Python bindings for M2Crypto, Python bindings for Requests, Python bindings for Sphinx, Python bindings for YAML, Python bindings for ZeroMQ, and MessagePack implementation by Python.

```
# apt install python-crypto python-jinja2 python-m2crypto python-requests python-sphinx python-yaml python-zmq msgpack-python
```

Install Cython.

```
# apt install cython
```

1d2) Clone git repository

Now, we must download the SaltStack source code. According to the official documentation, Calamari supports ***Salt 2014.7*** and does not work with newer versions.
SOURCE: http://calamari.readthedocs.io/en/latest/operations/minion_connect.html#install-salt-minion

The first place to look for this version is in the Official SaltStack repositories. Unfortunately the oldest version hosted by SaltStack(http://repo.saltstack.com) is ***2015.8*** as of now(***2017 March 17***). But SaltStack source code is hosted on GitHub and all versions of SaltStack is maintained there.

So we will use GitHub as our source for SaltStack.

Clone the ***2014.7*** branch of SaltStack Git repository.

```
# git clone https://github.com/saltstack/salt.git --branch=2014.7
```

This will clone the Git repository into directory ***salt***.

1d3) Build process

Change to the directory ***salt*** in terminal.

```
# cd salt
```

Build the Salt packages.

```
# dpkg-buildpackage
```

The build process will output following files in the directory ***/tmp/calamarirepo/***.

- salt_2014.7.1-1_amd64.changes **[Debian changes file]**
- salt_2014.7.1-1.dsc **[Debian Source Control file]**
- salt_2014.7.1-1.tar.xz **[Archived Upstream source tarball]**
- salt-api_2014.7.1-1_all.deb **[Debian package]**
- salt-cloud_2014.7.1-1_all.deb **[Debian package]**
- salt-common_2014.7.1-1_all.deb **[Debian package]**
- salt-doc_2014.7.1-1_all.deb **[Debian package]**
- salt-master_2014.7.1-1_all.deb **[Debian package]**
- salt-minion_2014.7.1-1_all.deb **[Debian package]**
- salt-ssh_2014.7.1-1_all.deb **[Debian package]**
- salt-syndic_2014.7.1-1_all.deb **[Debian package]**

But we only need the following files for installing SaltStack.

- salt-common_2014.7.1-1_all.deb
- salt-master_2014.7.1-1_all.deb
- salt-minion_2014.7.1-1_all.deb

2) Copying packages to Ceph nodes

Change to the directory ***/tmp/calamarirepo/*** in terminal.

```
# cd /tmp/calamarirepo
```

Copy the files as following

⟹ ***Calamari Server***, ***Calamari Clients***, ***Diamond***, and ***python-support*** packages to directory ***/root/*** of *cinder-ceph* node.

⟹ ***Salt Master***, ***Salt Minion***, and ***Salt Common*** packages to directory ***/root/*** of *cinder-ceph* node.

```
# scp calamari-server_1.0.0-1_amd64.deb calamari-clients.tar.gz diamond_3.1.0_all.deb python-support_1.0.15_all.deb root@10.3.0.3:/root
```

```
# scp salt-master_2014.7.1-1_all.deb salt-minion_2014.7.1-1_all.deb salt-common_2014.7.1-1_all.deb root@10.3.0.3:/root
```

Copy the files as following

⟹ ***Diamond*** and ***python-support*** packages to directory ***/root/*** of *ceph-osd1* node.

⟹ ***Salt Minion*** and ***Salt Common*** packages to directory ***/root/*** of *ceph-osd1* node.

```
# scp diamond_3.1.0_all.deb python-support_1.0.15_all.deb root@10.3.0.4:/root
```

```
# scp salt-minion_2014.7.1-1_all.deb salt-common_2014.7.1-1_all.deb root@10.3.0.4:/root
```

Copy the files as following

⟹ ***Diamond*** and ***python-support*** packages to directory ***/root/*** of *ceph-osd2* node.

⟹ ***Salt Minion*** and ***Salt Common*** packages to directory ***/root/*** of *ceph-osd2* node.

```
# scp diamond_3.1.0_all.deb python-support_1.0.15_all.deb root@10.3.0.5:/root
```

```
# scp salt-minion_2014.7.1-1_all.deb salt-common_2014.7.1-1_all.deb root@10.3.0.5:/root
```

3) Installing and configuring Calamari

For installing and Configuring Calamari, perform the following operations.

3a) Install and configure SaltStack

To install and configure SaltStack we must configure them separately in Ceph OSDs and admin nodes.

3a1) Configure Ceph admin node
3a2) Configure Ceph OSDs
3a3) Accept keys

3a1) Configure Ceph admin node

Let us install and configure both Salt Master and Minions in the *cinder-ceph* node.

First let us install Salt Master.

Change to the directory ***/root/*** in terminal.

```
root@cinder-ceph:~# cd /root
```

Install Python bindings for cryptographic algorithms and protocols, Python bindings for M2Crypto, Python bindings for YAML, YAML shared library, Python bindings for ZeroMQ, ZeroMQ shared library, Sodium crypto shared library, Debian control file format tools, and MessagePack implementation by Python.

```
root@cinder-ceph:~# apt install python-crypto python-m2crypto python-yaml libyaml-0-2 python-zmq libzmq5 libsodium18 dctrl-tools python-msgpack
```

Install Python bindings for systemd.

```
root@cinder-ceph:~# apt install python-systemd
```

NOTE:

The package python-systemd is not a package dependency for SaltStack 2014.7 packages, but a functional dependency.

Because SaltStack 2014.7 is aimed at Ubuntu 14.04 which uses Upstart, and not at Ubuntu 16.04 that uses systemd for service management.

In Ubuntu 16.04 without this package, we will see that Salt Master processes are running fine with no errors in Salt Master log file; but whenever we try to manage the Salt Master service using systemctl, it will timeout.

SaltStack 2015.8 packages available in Ubuntu 16.04 official repository has python-systemd as a package dependency.

Install the package python-support.

```
root@cinder-ceph:~# dpkg -i python-support_1.0.15_all.deb
```

Install Salt Common package.

```
root@cinder-ceph:~# dpkg -i salt-common_2014.7.1-1_all.deb
```

Install Salt Server package.

```
root@cinder-ceph:~# dpkg -i salt-master_2014.7.1-1_all.deb
```

Enable the Salt Master service.

```
root@cinder-ceph:~# systemctl enable salt-master
```

Start the Salt Master service.

```
root@cinder-ceph:~# systemctl start salt-master
```

Now we can install and configure Salt Minions.

Install Salt Minions package.

```
root@cinder-ceph:~# dpkg -i salt-minion_2014.7.1-1_all.deb
```

Configure Salt Minions.

In configuration file ***/etc/salt/minion***, add calamari master hostname

```
master: cinder-ceph.office.brookwin.com
```

Create a new configuration file ***/etc/salt/minion.d/calamari.conf*** with following content

```
master: cinder-ceph.office.brookwin.com
```

Enable the Salt Minion service.

```
root@cinder-ceph:~# systemctl enable salt-minion
```

Restart the Salt Minion service.

```
root@cinder-ceph:~# systemctl restart salt-minion
```

Check status of Salt Minion service.

```
root@cinder-ceph:~# systemctl status salt-minion
```

EXPECTED OUTPUT:

[ERROR] The Salt Master has cached the public key for this node, this salt minion will wait for 10 seconds before attempting to re-authenticate

3a2) Configure Ceph OSDs

Perform the following steps in Ceph OSDs *ceph-osd1* and *ceph-osd2*.

Let us install and configure Salt Minions.

Change to the directory ***/root/*** in terminal.

```
# cd /root
```

Install Python bindings for cryptographic algorithms and protocols, Python bindings for M2Crypto, Python bindings for YAML, YAML shared library, Python bindings for ZeroMQ, ZeroMQ shared library, Sodium crypto shared library, Debian control file format tools, and MessagePack implementation by Python.

```
# apt install python-crypto python-m2crypto python-yaml libyaml-0-2 python-zmq libzmq5 libsodium18 dctrl-tools python-msgpack
```

Install the package python-support.

```
# dpkg -i python-support_1.0.15_all.deb
```

Install Salt Common package.

```
# dpkg -i salt-common_2014.7.1-1_all.deb
```

Install Salt Minions package.

```
# dpkg -i salt-minion_2014.7.1-1_all.deb
```

Configure Salt Minions.

In configuration file ***/etc/salt/minion***, add calamari master hostname.

```
master: cinder-ceph.office.brookwin.com
```

Create a new configuration file ***/etc/salt/minion.d/calamari.conf*** with the following content

```
master: cinder-ceph.office.brookwin.com
```

Enable the Salt Minion service.

```
# systemctl enable salt-minion
```

Restart the Salt Minion service.

```
# systemctl restart salt-minion
```

Check status of Salt Minion service.

```
# systemctl status salt-minion
```

EXPECTED OUTPUT:

[ERROR] The Salt Master has cached the public key for this node, this salt minion will wait for 10 seconds before attempting to re-authenticate

3a3) Accept keys

Salt uses AES encryption for all communication between the Master and the Minion. This ensures that the commands sent to the Minions cannot be tampered with, and that communication between Master and Minion is authenticated through trusted, accepted keys. Before commands can be sent to a Minion, its key must be accepted on the Master.

List the keys known to Salt Master.

```
root@cinder-ceph:~# salt-key -L
```

EXPECTED OUTPUT:

```
Accepted Keys:
Unaccepted Keys:
ceph-osd1.office.brookwin.com
ceph-osd2.office.brookwin.com
cinder-ceph.office.brookwin.com
Rejected Keys:
```

Salt Master is aware of the three Minions, but none of the keys has been accepted.
To accept the keys and allow the Minions to be controlled by the Master, accept all the keys that are on Salt master.

```
root@cinder-ceph:~# salt-key -A
```

EXPECTED OUTPUT:

```
The following keys are going to be accepted:
Unaccepted Keys:
ceph-osd1.office.brookwin.com
ceph-osd2.office.brookwin.com
cinder-ceph.office.brookwin.com
Proceed? [n/Y]
```

Type ***Y*** and press Enter.

3b) Install and configure Diamond

Perform the following operations in all three Ceph Nodes.

Change to the directory ***/root/*** in terminal.

```
# cd /root
```

Install Python bindings for ConfigObj.

```
# apt install python-configobj
```

Install Diamond package.

```
# dpkg -i diamond_3.1.0_all.deb
```

Diamond has to be configured via the configuration file ***/etc/diamond/diamond.conf***. We will be saving this for later when Diamond configuration file will be populated during Calamari initialization process. So Diamond will be automatically configured during this event.

3c) Install and configure Calamari Clients

Create a directory ***/opt/calamari/webapp/content/***.

```
root@cinder-ceph:~# mkdir -p /opt/calamari/webapp/content
```

Change to the directory ***/opt/calamari/webapp/content/*** in terminal.

```
root@cinder-ceph:~# cd /opt/calamari/webapp/content
```

Extract the gzipped tarball ***calamari-clients.tar.gz*** to directory ***/opt/calamari/webapp/content/***.

```
root@cinder-ceph:~# tar -xzvf /root/calamari-clients.tar.gz
```

Calamari Client files are now installed in the directory ***/opt/calamari/webapp/content/***.

3d) Install and configure Calamari Server

Calamari Server has packages in the following four Application groups as dependencies.

Apache HTTP Server application group

⇒ Apache HTTP Server, Apache HTTP Server binary and modules, Apache HTTP Server common files, Apache HTTP Server utilities, Apache HTTP Server WSGI module, Apache Portable Runtime library, Apache Portable Runtime Utility shared library, SQLite3 Driver for Apache Portable Runtime Utility library, LDAP Driver for Apache Portable Runtime Utility library, Lua interpreter 5.1 shared library, Python 2.7 shared library

Package names

➠ *apache2, apache2-bin, apache2-data, apache2-utils, libapache2-mod-wsgi, libapr1, libaprutil1, libaprutil1-dbd-sqlite3, libaprutil1-ldap, liblua5.1-0, libpython2.7*

Cairo application group

⇒ Python bindings for Cairo, Cairo shared library, generic font configuration library runtime, generic font configuration library configuration, DejaVu fonts core, X C Binding render extension, X C Binding shm extension, X Rendering Extension client library

Package names

➠ *python-cairo*, *li bcairo2*, *libfontconfig1*, *fontconfig-config*, *fonts-dejavu-core*, *libxcb-render0*, *libxcb-shm0*, *libxrender1*

PostgreSQL application group

⇒ PostgreSQL supported, PostgreSQL 9.5 Server, PostgreSQL 9.5 Client, PostgreSQL Common, PostgreSQL client Common, PostgreSQL C client library, Simple debconf wrapper for OpenSSL

Package Names

➠ *postgresql*, *postgresql-9.5*, *postgresql-client-9.5*, *postgresql-common*, *postgresql-client-common*, *libpq5*, *ssl-cert*

Supervisor application group

⇒ Supervisor, Python bindings for meld3

Package Names

➠ *supervisor*, *python-meld3*

The above four Software groups are dependent on each other by package dependency. So we will have to install all these packages together. Execute the following command in terminal.

```
root@cinder-ceph:~#  apt install apache2 apache2-bin apache2-data apache2-utils libapache2-mod-wsgi libapr1 libaprutil1 libaprutil1-dbd-sqlite3 libaprutil1-ldap liblua5.1-0 libpython2.7 python-cairo libcairo2 libfontconfig1 fontconfig-config fonts-dejavu-core libxcb-render0 libxcb-shm0 libxrender1 postgresql postgresql-9.5 postgresql-client-9.5 postgresql-common postgresql-client-common libpq5 ssl-cert supervisor python-meld3
```

Change to the directory ***/root/*** in terminal.

```
root@cinder-ceph:~# cd /root
```

Install the Calamari Server package.

```
root@cinder-ceph:~# dpkg -i calamari-server_1.0.0-1_amd64.deb
```

EXPECTED OUTPUT:

ERROR: Module version does not exist!
salt-master: no process found

The above warnings do not affect the installation process of Calamari Server and can be safely ignored.
SOURCE:
http://access.redhat.com/documentation/en/red-hat-ceph-storage/1.3.2/paged/release-notes/chapter-4-known-issues
http://bugzilla.redhat.com/show_bug.cgi?id=1305133 *[Bug restricted for internal development process]*

We have an issue with Calamari Server's salt module ***ceph.py*** in directory ***/opt/calamari/salt/salt/_modules/***. After configuring Calamari, If we run certain salt queries, for example ***salt '*' ceph.get_heartbeats***, we will get the following error.

```
  File "/var/cache/salt/minion/extmods/modules/ceph.py", line 627, in cluster_status
    mds_epoch = status['mdsmap']['epoch']
KeyError: 'mdsmap'
```

This is because, for Ceph Jewel release, the ***`ceph status`*** JSON blob includes ***fsmap*** dict, instead of old ***mdsmap*** dict. The result of this error is that ***ceph status*** silently fails and Calamari thinks that there is no cluster configured.
SOURCE: http://github.com/ceph/calamari/commit/f911d6d503e4a1f28f795670be1a3e40e9143f3d

To fix this make the following changes.

In file ***/opt/calamari/salt/salt/_modules/ceph.py***, look for the below content.

```
mon_epoch = status['monmap']['epoch']
osd_epoch = status['osdmap']['osdmap']['epoch']
mds_epoch = status['mdsmap']['epoch']
```

Replace them with below content.

```
mon_epoch = status.get('monmap', {}).get('epoch')
osd_epoch = status.get('osdmap', {}).get('osdmap', {}).get('epoch')
mds_epoch = status.get('mdsmap', {}).get('epoch')
```

Restart the Salt master service.

```
root@cinder-ceph:~# systemctl restart salt-master
```

Restart the Salt minion service.

```
root@cinder-ceph:~# systemctl restart salt-minion
```

Wait for a minute. The salt module ***ceph.py*** will be cached from directory ***/opt/calamari/salt/salt/_modules/*** to directory ***/var/cache/salt/minion/extmods/modules/***.

Now let us check the user under which Apache HTTP Server is running.

```
root@cinder-ceph:~# ps -elF | grep apache2 | grep -v grep | grep -v root | awk '{print $3}' | uniq
```

EXPECTED OUTPUT:

```
www-data
```

In configuration file ***/etc/salt/master.d/calamari.conf***, comment out all Apache HTTP Server users except ***www-data***.

```
#  apache:
#    - log_tail.*
#  wwwrun:
#    - log_tail.*
```

Restart the Salt Master service.

```
root@cinder-ceph:~# systemctl restart salt-master
```

Finally, initialize the Calamari installation.

```
root@cinder-ceph:~# calamari-ctl initialize
```

- ✓ Leave the Username blank
- ✓ Enter a preferred email address
- ✓ Enter password as **101010**

EXPECTED OUTPUT:

```
[INFO] Loading configuration..
[INFO] Starting/enabling salt...
[INFO] Starting/enabling postgres...
[INFO] Initializing database...
[INFO] You will now be prompted for login details for the administrative user account.  This is
the account you will use to log into the web interface once setup is complete.
Username (leave blank to use 'root'):
Email address: [redacted]
Password:
Password (again):
Superuser created successfully.
[INFO] Initializing web interface...
[INFO] Starting/enabling services...
[INFO] Updating already connected nodes.
[INFO] Restarting services...
[INFO] Complete.
```

The Diamond instance will now be dead in all nodes. It should not have happened, but has happened any way. This could supposedly be a bug. Let us restart the diamond service in all nodes.

```
# systemctl restart diamond
```

Now, let us check the status of carbon-cache process.

```
root@cinder-ceph:~# supervisorctl status | grep carbon-cache
```

EXPECTED OUTPUT:

```
carbon-cache            BACKOFF   can't find command '/opt/calamari/venv/bin/carbon-cache.py'
```

This happens because we had skipped the installation of Carbon during Calamari Server build process. Carbon is still missing from the Calamari Server Virtual environment. Let us install Carbon in our Calamari Server Virtual environment.

```
root@cinder-ceph:~# /opt/calamari/venv/bin/pip2.7 install carbon --install-option="--prefix=/opt/calamari/venv" --install-option="--install-lib=/opt/calamari/venv/lib/python2.7/site-packages"
```

This will install ***carbon 0.9.15*** with python bindings for following package versions.

- ***txAMQP 0.6.2*** (http://github.com/txamqp/txamqp)
- ***zope.interface 4.3.3*** (http://github.com/zopefoundation/zope.interface)

Restart the supervisor service.

```
root@cinder-ceph:~# systemctl restart supervisor
```

Enable supervisor service.

```
root@cinder-ceph:~# systemctl enable supervisor
```

4) Check for errors

Finally, let us check if there are any errors in our deployment of Calamari.

Perform the following operations in all nodes.

Copy the directory ***/var/log/*** as a backup renamed with current date and time, delete any existing log files from directory ***/var/log/***, and reboot the server.

```
# cp -avr /var/log /var/log_bak_$(date +"%Y%m%d-%H%M%S") && find /var/log -type f | xargs rm -f && reboot
```

After the server comes up, check for any critical messages or errors or warnings.

```
# grep crit -irl /var/log
```

```
# grep err -irl /var/log
```

```
# grep warn -irl /var/log
```

5) Starting and running Calamari

Let us check if our Calamari installation can properly communicate with the Ceph cluster.

Use the execution module ***test.ping*** to list all the connected Ceph nodes.

```
root@cinder-ceph: ~#  salt '*' test.ping
```

EXPECTED OUTPUT:

```
cinder-ceph.office.brookwin.com:
  True
ceph-osd1.office.brookwin.com:
  True
ceph-osd2.office.brookwin.com:
  True
```

Ping using execution module ***test.ping*** is not an ICMP ping.

Use the module ***ceph.heartbeat*** to list the heartbeats coming from all Ceph nodes.

```
root@cinder-ceph: ~#  salt '*' ceph.heartbeat
```

EXPECTED OUTPUT:

```
ceph-osd1.office.brookwin.com:
  None
cinder-ceph.office.brookwin.com:
  None
ceph-osd2.office.brookwin.com:
  None
```

Use the module ***ceph.get_heartbeats*** to query Ceph cluster and get cluster information.

```
root@cinder-ceph: ~#  salt '*' ceph.get_heartbeats
```

EXPECTED OUTPUT:

(Click here)

Use the module ***ceph.get_cluster_object***, to retrieve the full copy of the cluster map in all nodes.

```
root@cinder-ceph: ~#  salt '*' ceph.get_cluster_object ceph health None
```

EXPECTED OUTPUT:

(Click here)

Finally we can access our Calamari installation using following login details.

URL: *http://10.3.0.3*
Username: *root*
Password: *101010*

Further we can view the details of Ceph cluster using Django REST framework API.

http://10.3.0.3/api/v1/
http://10.3.0.3/api/v1/cluster
http://10.3.0.3/api/v1/user

http://10.3.0.3/api/v2/
http://10.3.0.3/api/v2/cluster
http://10.3.0.3/api/v2/user
http://10.3.0.3/api/v2/server

Done.