
gdeploy Documentation

Release 1

Nandaja Varma

Nov 30, 2017

Contents

1	Quick links	3
2	Contents	5
2.1	Installation	5
2.1.1	Prerequisites	5
2.1.2	Installing Ansible	5
2.1.3	Installing gdeploy	5
2.2	Getting Started	6
2.2.1	Invoking gdeploy	7
2.2.2	Writing configuration file for gdeploy	7
2.2.3	Invoking gdeploy	7
2.3	Usage	8
2.4	Debugging	8
2.5	Configuration file format	8
2.6	Features	9
2.7	Maintainer	10
2.8	Developer Documentation	10
2.8.1	gdeploy - Developer setup	10
2.8.2	gdeploy architecture	10
2.8.3	Adding your first feature to gdeploy	11
2.8.4	Testing your code	12
2.8.5	Guidelines for contribution	12
2.9	Design	12
2.10	Testsuite	12
2.11	Frequently Asked Questions	12
2.11.1	Why do we need gdeploy, when Ansible is available?	12
2.11.2	How does gdeploy help in setting up GlusterFS clusters?	12
2.11.3	Does gdeploy help in installing GlusterFS packages?	12
2.11.4	Is gdeploy only for installing and deploying GlusterFS?	12
2.11.5	Can I run arbitrary scripts using gdeploy?	13
2.11.6	My gdeploy run is failing with Module Error, why?	13
2.12	Examples	13
2.12.1	Using gdeploy to create a 1x3 Gluster Volume	13
2.12.2	Using gdeploy to create 2x2 gluster volume	15
2.12.3	Write a config file to do the backend setup	17
2.12.4	How to start and stop services	19

2.12.5	Set/unset volume options on an existing volume	21
2.12.6	Setting the options on an existing volume	21
2.12.7	Resetting the options on the existing volume	23
2.12.8	Installing packages from yum repositories	23
2.12.9	How to disable repos	24
2.12.10	Quota setup on an existing volume	25
2.12.11	Enabling and disabling quota	26
2.12.12	Create a Gluster volume and set up quota	29
2.12.13	Set a 5GB limit on a directory using quota	35
2.12.14	Creating a volume and setting a tuning profile on it	37
2.12.15	Setting a tuning profile on an existing volume	38
2.12.16	NFS Ganesha setup end-to-end	38
2.12.17	Unexporting a volume and destroying an NFS-Ganesha HA Cluster	39

3 Indices and tables 41

gdeploy is an [Ansible](#) based deployment tool. Initially gdeploy was written to install GlusterFS clusters, eventually it grew out to do lot of other things. On a given set of hosts, gdeploy can create physical volumes, volume groups, and logical volumes, install packages, subscribe to RHN channels, run shell commands, create GlusterFS volumes and lot more.

CHAPTER 1

Quick links

- [Git repositories](#)
- [Mailing list](#)

2.1 Installation

2.1.1 Prerequisites

gdeploy requires the following packages:

- Python 2.x
- Ansible >= 1.9.x
- python-argparse
- PyYAML
- Jinja2

2.1.2 Installing Ansible

Follow instructions in the Ansible documentation on how to install Ansible, which can be found [here](#).

2.1.3 Installing gdeploy

gdeploy can be installed using pre-built RPM or can be installed from source.

Installing from RPM

Latest version of gdeploy RPMs can be downloaded from [here](#) and installed

```
Using yum:  
$ sudo yum install ./gdeploy-<version>-<release>.rpm
```

```
Using dnf:  
$ sudo dnf install ./gdeploy-<version>-<release>.rpm
```

Installing from source

Alternatively gdeploy can be installed from source

```
$ git clone git@github.com:gluster/gdeploy.git  
$ cd gdeploy
```

Make sure you have gcc and python-devel installed

```
$ sudo yum install gcc python-devel redhat-rpm-config  
$ sudo pip install -r requirements.txt
```

Setup gdeploy

Run the gdeploy_setup.sh file from the root directory of gdeploy

```
$ cd gdeploy  
$ sudo ./gdeploy_setup.sh
```

OR Setup manually as follows

1. Add ansible modules to ANSIBLE_LIBRARY environment variable

```
$ echo "export ANSIBLE_LIBRARY=$ANSIBLE_LIBRARY:path/to/gdeploy/modules/" >> ~/.  
↪bashrc
```

‘path/to’ will be replaced by the path on your system on which gdeploy is installed.

2. Add ansible playbooks(inside the templates directory) to GDEPLOY_TEMPLATES environment variable

```
$ echo "export GDEPLOY_TEMPLATES='path/to/gdeploy'" >> ~/.bashrc
```

‘path/to’ will be replaced by the path on your system on which gdeploy is installed.

```
$ source ~/.bashrc
```

3. Install gdeploy using setuptools

```
$ sudo python setup.py install
```

2.2 Getting Started

gdeploy works by interacting with the remote nodes by communicating via passwordless ssh connections. Passwordless ssh connections have to be created to all the nodes on which gdeploy is going to create and configure a Gluster volume.

To setup passwordless ssh to the nodes, follow the steps below:

- Generate passphrase-less ssh-keys for the nodes which are to be used with gdeploy, running the following command:

```
$ ssh-keygen -t rsa -N ''
```

- Set up passwordless ssh access between the node running gdeploy and servers by running the following command:

```
$ ssh-copy-id root@hostname
```

'hostname' refers to the unique IP address of the node.

You would have to run these commands for all the nodes.

Sometimes, you may encounter a "Connection Refused" error. In this case, you need to check whether the ssh service is running on your system. You may use this command to check the same:

```
$ systemctl status sshd.service
```

If the service is not running, use this command to start the service:

```
$ systemctl start sshd.service
```

Once ssh connections to all the nodes are established, we can start writing a configuration file.

That's it! Now the machines are ready to be used with gdeploy.

2.2.1 Invoking gdeploy

gdeploy needs a configuration file to run. Write a configuration file, see [Writing configuration file for gdeploy](#) section below for more details.

Invoke gdeploy with configuration file as an argument:

```
$ gdeploy -c sample.conf
```

2.2.2 Writing configuration file for gdeploy

gdeploy configuration file is a plain text file comprising multiple sections, the sections are arranged in an order based on what needs to be achieved.

A very simple configuration file named enable-ntpd.conf which enables and starts ntpd on all the nodes looks like:

```
[hosts]
10.0.0.1
10.0.0.2
10.0.0.3

[service1]
action=enable
service=ntpd

[service2]
action=start
service=ntpd
```

2.2.3 Invoking gdeploy

Invoke gdeploy with configuration file as an argument:

```
$ gdeploy -c sample.conf
```

The configuration file given above will enable and start ntpd on three nodes. 10.0.0.1, 10.0.0.2, and 10.0.0.3 when the following command is invoked:

```
$ gdeploy -c enable-ntpd.conf
```

INFO: The 'ntp' package has to be installed on both the nodes in order for this configuration file to run. This can be done using the command “dnf install ntp”.

For more details on the list of all the features supported by gdeploy, refer *gdeploy features* topic.

2.3 Usage

gdeploy needs a configuration file to do anything useful. Refer *Configuration file format* for an example.

gdeploy -h will list the available options for gdeploy:

```
$ gdeploy -h
usage: gdeploy [-h] [-v] [-vv] [-c CONFIG_FILE] [-k]

optional arguments:
  -h, --help            show this help message and exit
  --version            show program's version number and exit
  -c CONFIG_FILE        Configuration file
  -k                    Keep the generated ansible utility files
  --trace              Turn on the trace messages in logs
  -vv                  verbose mode
  --addfeature FEATURE_NAME
                        Add new feature to gdeploy
```

gdeploy --addfeature FEATURE_NAME will create a skeleton to add a new feature to gdeploy. For more details on how to write a feature refer *Developer Documentation*.

Invoke gdeploy with configuration file as an argument:

```
$ gdeploy -c config-file
```

More example configuration files can be found [here](#).

2.4 Debugging

2.5 Configuration file format

This section explains the gdeploy configuration file format. There is no rule to name the configuration files in gdeploy as long as the file name is mentioned with -c option for gdeploy. For example gluster.conf, gluster, gluster.txt, gluster.config are all valid names for a gdeploy configuration file.

gdeploy configuration file is split into two parts.

1. Inventory
2. Features

1. Inventory

The section is named [hosts], this is a mandatory section, hosts that are to be configured have to be listed in this section. Hostnames or ip addresses have to be listed one per line. For example:

```
[hosts]
10.0.0.1
10.0.0.2
10.0.0.3
```

2. Features

There can be more than one feature listed in the configuration file, each separated by a newline. Every feature section has one or more variables which controls how the feature is configured/deployed. The below example has two features, firewallld and service that will be configured on all the hosts listed in the [hosts] section:

```
[hosts]
10.0.0.1
10.0.0.2
10.0.0.3
10.0.0.4

[firewalld]
action=add
ports=111/tcp,2049/tcp,54321/tcp
permanent=true
zone=public

[service1]
action=enable
service=ntpd

[service2]
action=restart
service=ntpd
```

If a feature has to be used more than once, then it has to be in different sections and numbered to make it unique as shown in the above example.

The list of available features and their complete documentation can be found in [Features](#) page.

2.6 Features

gdeploy supports a number of features, each feature does a particular task. For example setting up lvm and creating filesystem, subscribe to Red Hat Subscription Network, creating GlusterFS volumes, so on.

Each feature provides variables that can be set to fine tune the system. This document explains the available features and their tunable variables.

1. rst_gdeployclients
2. rst_gdeployfirewalld
3. rst_gdeploylvm
4. rst_nfsganesha
5. rst_gdeploypeer

6. rst_gdeployquota
7. rst_sambaetdb
8. rst_gdeployscript
9. rst_gdeployservice
10. rst_gdeployshell
11. rst_gdeploysnapshot
12. rst_gdeployssl
13. rst_subscriptionmanagement
14. rst_gdeploysystemd
15. rst_gdeployupdatefile
16. rst_glusterfsvolume
17. rst_gdeployyum

2.7 Maintainer

2.8 Developer Documentation

With gdeploy, the user get to remotely configure a lot of features in multiple machines, without worrying dealing with the complexities of writing Ansible Playbooks, modules, etc. It is the duty of the developer to worry about all these things. Adding a new feature to gdeploy is relatively easy, but comes with the cost of writing playbooks and modules(if necessary) for Ansible. So this guide assumes that the developer is comfortable with Python and has got the basic working knowledge of Ansible.

2.8.1 gdeploy - Developer setup

To setup the development environment, refer *Installing from source*.

Proceed further once you are done with the setup.

2.8.2 gdeploy architecture

gdeploy is a very lightweight, simple tool that efficiently make use of Ansible, hiding the major complexities of it from the user, that does numerous operations on remote machines sequentially. To do this, gdeploy divides the logic to 3 parts:

1. Ansible module that implements the desired logic to be executed in the remote machine.
2. Ansible playbook that makes use of the already present Ansible module and specifies how a particular operation is to be performed.
3. A section independent from both the above sections that reads the user configuration file, parses it accordingly, sets default values for some if not provided by the user, and then populate all these data in variables that the Ansible playbook will then use. More about these variable can be found in the [Ansible documentation about them](#)

For this, gdeploy provide what we call the sections or features(*Features*). Each feature will have n-number of options for which the user will specify the value.

2.8.3 Adding your first feature to gdeploy

In order to add a new feature to Ansible, the developer has to make sure the three above mentioned components or sections are written properly and works as intended.

The development process can be started in the following order:

1. Write the Ansible module, **if** needed.
2. Write Ansible playbooks that does the necessary operations **in** the right order.
3. Add a feature **with** a suitable name matching your requirement to the gdeploy framework.

**** Add a feature to gdeploy ****

Use the gdeploy option `--addfeature` for this. To add a feature names “myfeature”:

```
$ gdeploy --addfeature myfeature
```

This will create a directory `<l>myfeature</l>` under `<l>features</l>` directory. If you look inside this directory, you will see that gdeploy has created 3 files for you: an init file, a JSON file and a python script. You need just edit the JSON file and the python script to make gdeploy do what you want. The JSON file is used by gdeploy to validate the user configuration provided for your feature(in this case, ‘myfeature’). The necessary option for every feature in gdeploy is the option named ‘action’. Specify each one of your feature’s action inside the action hash in the JSON file. Each of these action keys will have a list names ‘options’ which will specify the options that is to be provided corresponding to each of these actions. [This](#) is the JSON file written to implement the snapshot feature in gdeploy.

Once your JSON is ready, the next big task is to create playbooks to run for each of these actions. This is where we cannot help you much. Writing playbooks and modules depends on your feature. So put your Python and Ansible skills to good use and write some cool playbooks. Playbooks should go under the directory `<l>playbooks</l>` under the working directory and modules should go under the directory `<l>modules</l>` under the working directory. Once your playbooks are in, add these playbook file names to the file `defaults.py`, just because it is cleaner.

Now you just have to let gdeploy know which playbook corresponds to which feature action. This is where the python script comes into picture. There should be a function corresponding to each feature action inside this Python script. The function name should be in the format ‘myfeature_actionname’. You need just edit the dummy method names provided inside the script. I am sure you will figure it out. It is pretty straight forward. As you will see inside the scripts, each function will have a parameter being passed, ‘section_dict’. This dictionary holds the as keys and values, the options and their corresponding values provided by the user in her configuration file under the section ‘myfeature’. Just print it out and see for yourself if you are happy with the format. You can modify the keys and values in the dictionary as per your needs. Each function should return 2 parameters: One is the modified or not modified section_dict and other is the playbook to be run for that particular section. Just edit the YML_NAME and let the defaults be.

2.8.4 Testing your code

2.8.5 Guidelines for contribution

2.9 Design

2.10 Testsuite

2.11 Frequently Asked Questions

1. *Why do we need gdeploy, when Ansible is available?*
2. *How does gdeploy help in setting up GlusterFS clusters?*
3. *Does gdeploy help in installing GlusterFS packages?*
4. *Is gdeploy only for installing and deploying GlusterFS?*
5. *Can I run arbitrary scripts using gdeploy?*
6. *My gdeploy run is failing with Module Error, why?*

2.11.1 Why do we need gdeploy, when Ansible is available?

gdeploy enables configuration and provisioning of GlusterFS and the file access protocols using configurations and tunables which are tested and recommended by the maintainers. This enables a system administrator to have an easy way to create consistent and repeatable deployment paths.

2.11.2 How does gdeploy help in setting up GlusterFS clusters?

Installation, configuration and provisioning of a GlusterFS deployment involves a sequence of steps to be executed in the proper order. This would include deployment-specific detail such as:

1. Setting up PV, VG, LV (thinpools if necessary).
2. Peer probing the nodes.
3. Using the CLI based volume creation steps

gdeploy provides a simple way to complete the steps and include specifics such as configuring volume options and such.

2.11.3 Does gdeploy help in installing GlusterFS packages?

gdeploy has a configuration workflow design which enables it to be used for package installation, either from upstream builds or, from a specific vendor provided content distribution mechanism viz. Red Hat's CDN

2.11.4 Is gdeploy only for installing and deploying GlusterFS?

While gdeploy is intended to streamline the administrator experience during installation and deployment of GlusterFS, it can be used to install other packages, custom scripts and modules for configuration. The hc.conf is an example of how gdeploy can enable the set-up and configuration for a HyperConverged stack using GlusterFS.

Refer [hc.conf](#) for an example for things gdeploy can achieve.

2.11.5 Can I run arbitrary scripts using gdeploy?

Yes. Scripts which aid and extend the deployment setup can be configured to run from gdeploy.

See the *script* module. Refer [hc.conf](#) for an example for script module usage.

2.11.6 My gdeploy run is failing with Module Error, why?

The error is due to the Ansible version installed. This is possibly because you might be using Ansible 2.0. gdeploy currently supports 1.9.x versions of Ansible.

2.12 Examples

2.12.1 Using gdeploy to create a 1x3 Gluster Volume

To create 1*3 gluster volume we would need three bricks which may or may not be on the same machine. It is recommended that these three bricks reside on different machines.

Step 1:

Create the following configuration file:

```
[hosts]
10.70.43.127
10.70.42.190
10.70.42.232

[backend-setup]
devices=/dev/vdb
vgs=1_3_gluster
pools=pool1
lvs=lv2
mountpoints=/mnt/data1
brick_dirs=/mnt/data1/1

[peer]
manage=probe

[volume]
action=create
volname=volumel
replica=yes
replica_count=3
force=yes

[clients]
action=mount
volname=volumel
hosts=192.168.122.19
fstype=glusterfs
client_mount_points=/mnt/client_mount
```

Step 2:

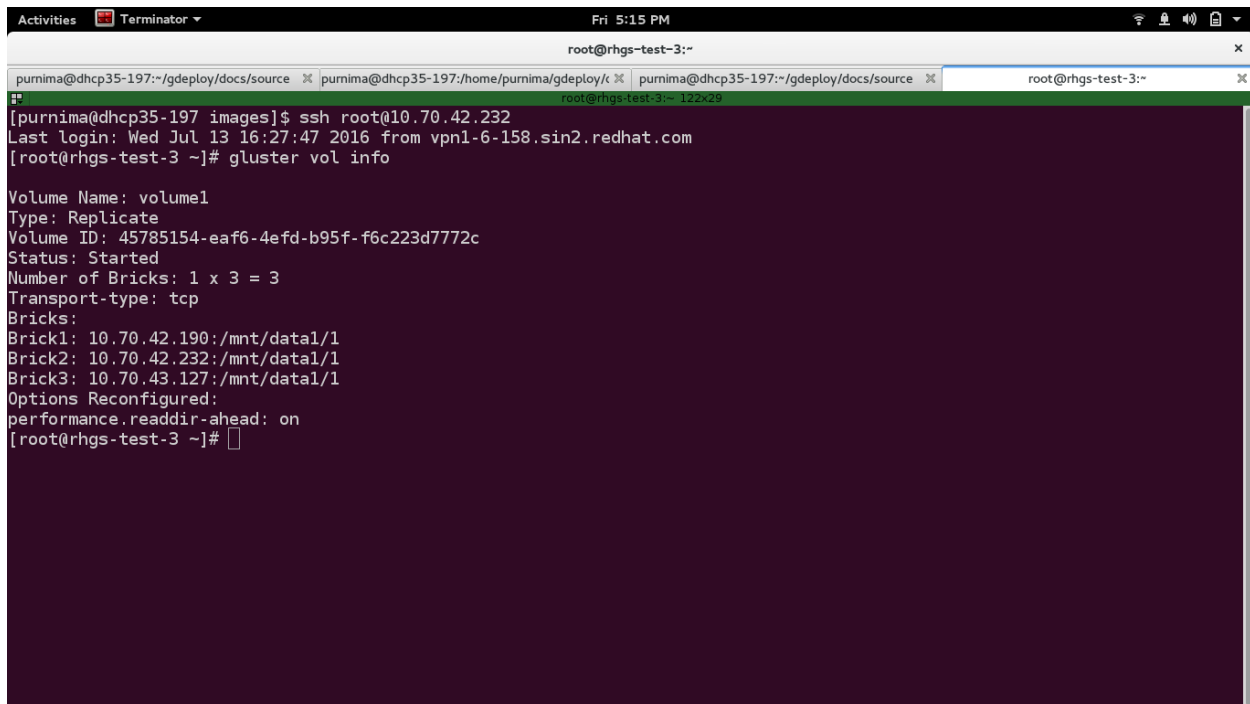
Save the file by giving it some name e.g. '1_3_volume.conf'. Invoke gdeploy and run the file using:

```
$gdeploy -c 1_3_volume.conf
```

Step 3:

Check whether a gluster volume has been created by running the below command:

```
$gluster vol info
```

A screenshot of a terminal window titled 'Terminator' showing a multi-tabbed environment. The active tab is 'root@rhgs-test-3:~'. The terminal shows the user 'purnima@dhcp35-197' logging in via SSH to 'root@10.70.42.232'. The user then runs the command 'gluster vol info'. The output displays details for 'Volume Name: volume1', including its type (Replicate), ID, status (Started), number of bricks (1 x 3 = 3), transport type (tcp), and the list of bricks (10.70.42.190:/mnt/data1/1, 10.70.42.232:/mnt/data1/1, 10.70.43.127:/mnt/data1/1). It also shows options reconfigured, such as 'performance.readdir-ahead: on'.

```
Activities Terminator Fri 5:15 PM
root@rhgs-test-3:~
purnima@dhcp35-197:~/gdeploy/docs/source x purnima@dhcp35-197:/home/purnima/gdeploy/c x purnima@dhcp35-197:~/gdeploy/docs/source x root@rhgs-test-3:~
[purnima@dhcp35-197 images]$ ssh root@10.70.42.232
Last login: Wed Jul 13 16:27:47 2016 from vpn1-6-158.sin2.redhat.com
[root@rhgs-test-3 ~]# gluster vol info

Volume Name: volume1
Type: Replicate
Volume ID: 45785154-eaf6-4efd-b95f-f6c223d7772c
Status: Started
Number of Bricks: 1 x 3 = 3
Transport-type: tcp
Bricks:
Brick1: 10.70.42.190:/mnt/data1/1
Brick2: 10.70.42.232:/mnt/data1/1
Brick3: 10.70.43.127:/mnt/data1/1
Options Reconfigured:
performance.readdir-ahead: on
[root@rhgs-test-3 ~]#
```

Step 4:

Now you can start writing to the volume using your client machine (192.168.122.19 in our case) by traversing to the path you have mentioned under "client_mount" using the following command:

```
$ sudo touch f1 f2 f3
```

This command will create three files under the directory /mnt/client_mount

You can also check whether the files have been created and replicated thrice inside the directory /mnt/data1/1 on the remote nodes by running the command:

```
$ ls
```

```

Activities Terminator Tue 15:43
root@rhgs-test-4:/mnt/data1/1

poo@poo:~/config root@rhgs-test-4:/mnt/data1/1 poo@poo:~/client_mount
root@rhgs-test-2:/mnt/data1/1 49x15 root@rhgs-test-1:/mnt/data1/1 49x15
[poo@poo ~]$ ssh root@10.70.42.190
Last login: Sun Jul 10 13:23:50 2016 from vpn1-5-242.sin2.redhat.com
[root@rhgs-test-2 ~]# cd /mnt/data1/1
[root@rhgs-test-2 1]# ls
[root@rhgs-test-2 1]# ls
f1 f2 f3
[root@rhgs-test-2 1]#

[poo@poo ~]$ ssh root@10.70.43.127
Last login: Sun Jul 10 13:01:32 2016 from vpn1-5-242.sin2.redhat.com
[root@rhgs-test-1 ~]# cd /mnt/data/1
-bash: cd: /mnt/data/1: No such file or directory
[root@rhgs-test-1 ~]# cd /mnt/data1/1
[root@rhgs-test-1 1]# ls
f1 f2 f3
[root@rhgs-test-1 1]#

root@rhgs-test-3:/mnt/data1/1 49x15
[poo@poo ~]$ ssh root@10.70.42.232
Connection closed by 10.70.42.232
[poo@poo ~]$
[poo@poo ~]$
[poo@poo ~]$
[poo@poo ~]$ ssh root@10.70.42.232
Last login: Sun Jul 10 13:02:12 2016 from vpn1-5-242.sin2.redhat.com
[root@rhgs-test-3 ~]# cd /mnt/data1/1
[root@rhgs-test-3 1]# ls
f1 f2 f3
[root@rhgs-test-3 1]#

```

We can see that the files have been successfully replicated on all the three nodes.

2.12.2 Using gdeploy to create 2x2 gluster volume

To create 2x2 gluster volume, you would need four bricks which may or may not be on the same machine. It is recommended that these four bricks reside on different machines.

Step 1:

Create the following configuration file:

```

[hosts]
10.70.43.127
10.70.42.190
10.70.42.232
10.70.43.67

[backend-setup]
devices=/dev/vdb
mountpoints=/gluster/brick1
brick_dirs=/gluster/brick1/one

[volume]
action=create
volname=sample_volume

```

```
replica=yes
replica_count=2
force=yes

[clients]
action=mount
hosts=192.168.122.19
fstype=glusterfs
client_mount_points=/mnt/random_client
```

Step 2:

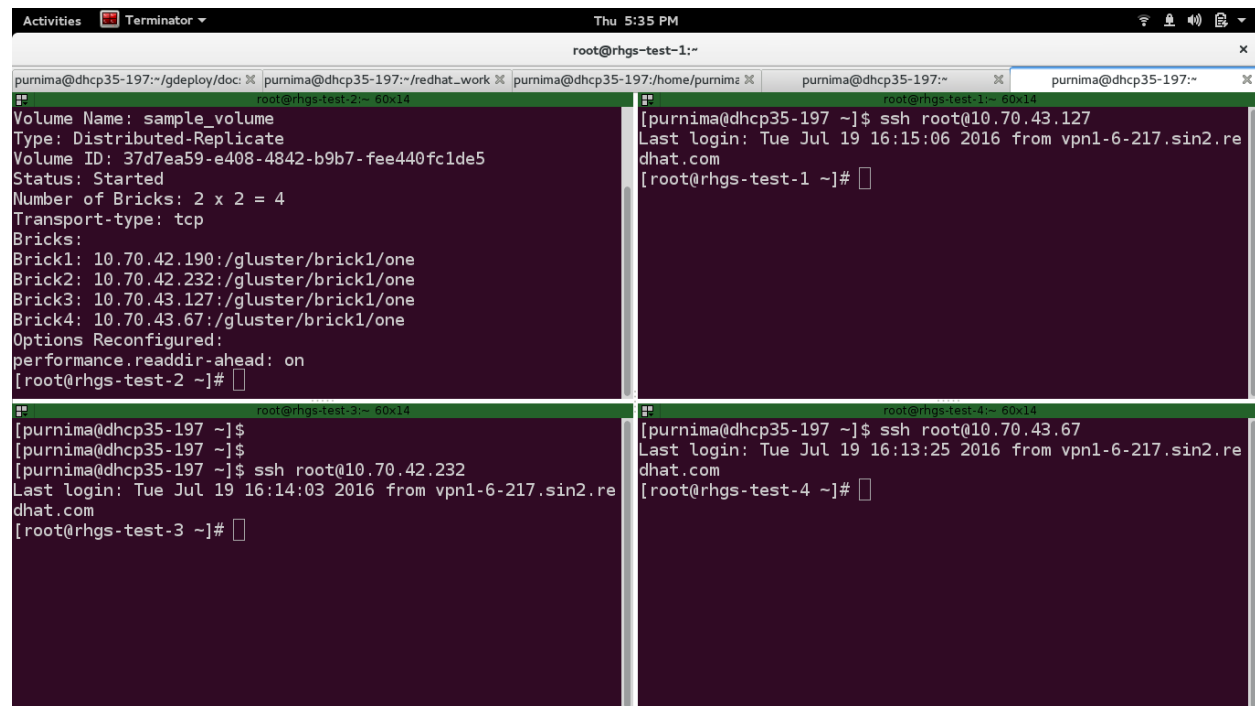
Save the file by giving it some name e.g. ‘2x2-gluster-volume.conf’. Invoke gdeploy and run the file using:

```
$ gdeploy -c 2x2-gluster-volume.conf
```

Step 3:

To check whether a gluster volume has been created by running the below command:

```
$ gluster vol info
```



The screenshot shows a terminal window with the following content:

```
root@rhgs-test-1:~# gluster vol info
Volume Name: sample_volume
Type: Distributed-Replicate
Volume ID: 37d7ea59-e408-4842-b9b7-fee440fc1de5
Status: Started
Number of Bricks: 2 x 2 = 4
Transport-type: tcp
Bricks:
Brick1: 10.70.42.190:/gluster/brick1/one
Brick2: 10.70.42.232:/gluster/brick1/one
Brick3: 10.70.43.127:/gluster/brick1/one
Brick4: 10.70.43.67:/gluster/brick1/one
Options Reconfigured:
performance.readdir-ahead: on
[root@rhgs-test-2 ~]#
```

Below this, there are four terminal windows showing SSH connections to remote nodes:

```
[purnima@dhcp35-197 ~]$ ssh root@10.70.43.127
Last login: Tue Jul 19 16:15:06 2016 from vpn1-6-217.sin2.re
dhat.com
[root@rhgs-test-1 ~]#
```

```
[purnima@dhcp35-197 ~]$ ssh root@10.70.43.67
Last login: Tue Jul 19 16:13:25 2016 from vpn1-6-217.sin2.re
dhat.com
[root@rhgs-test-4 ~]#
```

```
[purnima@dhcp35-197 ~]$ ssh root@10.70.42.232
Last login: Tue Jul 19 16:14:03 2016 from vpn1-6-217.sin2.re
dhat.com
[root@rhgs-test-3 ~]#
```

Step 3:

Now you can start writing to the volume using your client machine (192.168.122.19 in our case) by traversing to the path you have mentioned under “random_client” using the following command:

```
$ sudo touch 1 2 3 4 5
```

This command will create five files under the directory /home/poo/random_client.

You can also check whether the files have been created and replicated thrice inside the directory /gluster/brick1/one on the remote nodes by running the command:

```
$ ls
```

```

root@rhgs-test-4:/gluster/brick1/one
purnima@dhcp35-197:~/gdeploy/doc$ purnima@dhcp35-197:~/redhat_work$ purnima@dhcp35-197:/home/purnima$ purnima@dhcp35-197:~$ purnima@dhcp35-197:~$
root@rhgs-test-4:/gluster/brick1/one 60x14
Status: Started
Number of Bricks: 2 x 2 = 4
Transport-type: tcp
Bricks:
Brick1: 10.70.42.190:/gluster/brick1/one
Brick2: 10.70.42.232:/gluster/brick1/one
Brick3: 10.70.43.127:/gluster/brick1/one
Brick4: 10.70.43.67:/gluster/brick1/one
Options Reconfigured:
performance.readdir-ahead: on
[root@rhgs-test-2 ~]# cd /gluster/brick1/one
[root@rhgs-test-2 one]# ls
1 5
[root@rhgs-test-2 one]#

purnima@dhcp35-197 ~]$ ssh root@10.70.43.127
Last login: Tue Jul 19 16:15:06 2016 from vpn1-6-217.sin2.re
dhat.com
[root@rhgs-test-1 ~]# cd /gluster/brick1/one
[root@rhgs-test-1 one]# ls
2 3 4
[root@rhgs-test-1 one]#

purnima@dhcp35-197 ~]$ ssh root@10.70.42.232
Last login: Tue Jul 19 16:14:03 2016 from vpn1-6-217.sin2.re
dhat.com
[root@rhgs-test-3 ~]# cd /gluster/brick1/one
[root@rhgs-test-3 one]# ls
1 5
[root@rhgs-test-3 one]#

purnima@dhcp35-197 ~]$ ssh root@10.70.43.67
Last login: Tue Jul 19 16:13:25 2016 from vpn1-6-217.sin2.re
dhat.com
[root@rhgs-test-4 ~]# cd /gluster/brick1/one
[root@rhgs-test-4 one]# ls
2 3 4
[root@rhgs-test-4 one]#

```

We can see that the files have been successfully replicated on all the four nodes.

2.12.3 Write a config file to do the backend setup

[backend-setup] section is used to configure the disks on all the hosts mentioned in the [hosts] section. If the disk names vary from host to host then [backend-setup:<hostname>:<ip>] can be used to do setup backend on the particular host.

Step 1:

Create an empty file and give it any arbitrary name and add the following lines to it:

```

# This is a mandatory section, and hostnames/ip-address are listed one per line.

[hosts]
10.70.43.127
10.70.42.190
10.70.42.232
10.70.43.67

# Backend setup for all the hosts listed inside [hosts] section

[backend-setup]
devices=/dev/vdb
mountpoints=/gluster/brick1
brick_dirs=/gluster/brick1/one

# Backend setup for 10.70.46.77 with default gdeploy generated names for
# Volume Groups and Logical Volumes. Volume names will be GLUSTER_vg1,
# GLUSTER_vg2...
#
# [backend-setup:10.70.43.127]

```

```
# devices=vdb

# Backend setup for remaining 3 hosts in the `hosts` section with custom names
# for Volumes Groups and Logical Volumes.
#
# [backend-setup:10.70.46.{130,32,110}]
# devices=vdb,vdc,vdd
# vgs=vg1,vg2,vg3
# pools=pool1,pool2,pool3
# lvs=lv1,lv2,lv3
# mountpoints=/mnt/data1,/mnt/data2,/mnt/data3
# brick_dirs=/mnt/data1/1,/mnt/data2/2,/mnt/data3/3
```

Step 2:

Invoke gdeploy and run the file using:

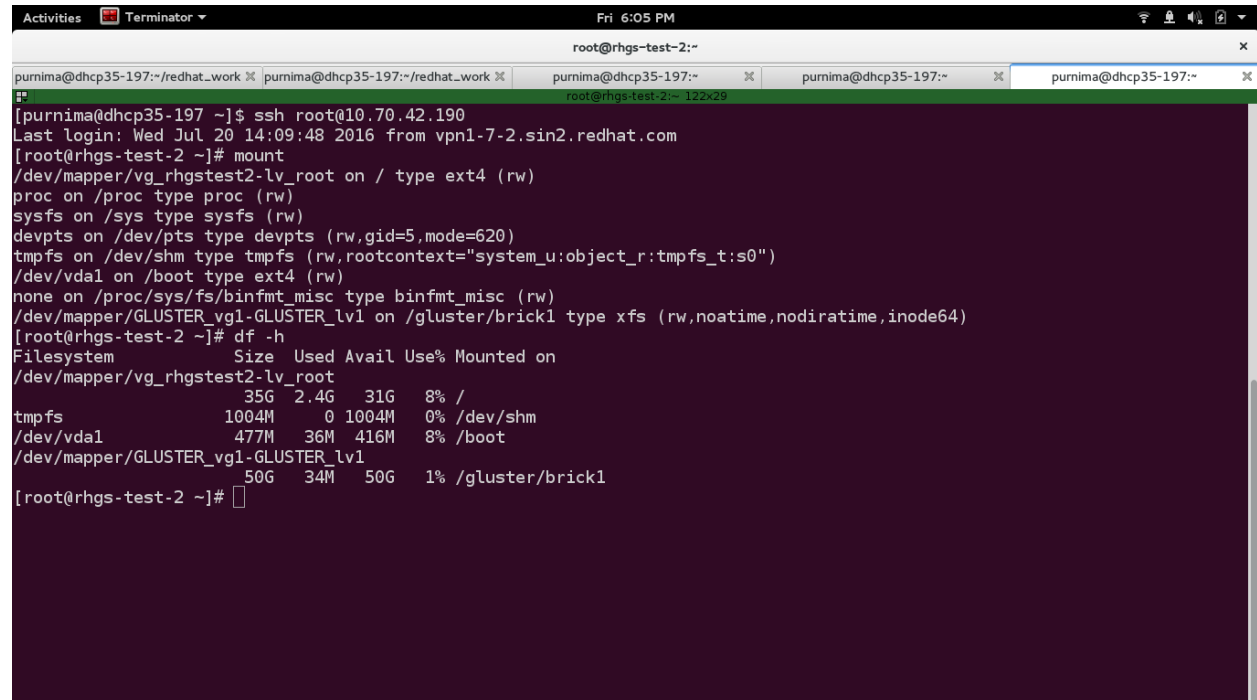
```
$ gdeploy -c backend-setup.conf
```

Step 3:

To see if the GLUSTER_vg1 (which is the default name for gluster volume group) has been mounted on the desired directory or not. You can either run `mount` or `df -h`:

```
$ mount
```

You'll see something like this.



```
Activities Terminator Fri 6:05 PM
root@rhgs-test-2:~
purnima@dhcp35-197:~/redhat_work x purnima@dhcp35-197:~/redhat_work x purnima@dhcp35-197:~ x purnima@dhcp35-197:~ x
purnima@dhcp35-197:~$ ssh root@10.70.42.190
Last login: Wed Jul 20 14:09:48 2016 from vpn1-7-2.sin2.redhat.com
[root@rhgs-test-2 ~]# mount
/dev/mapper/vg_rhgstest2-lv_root on / type ext4 (rw)
proc on /proc type proc (rw)
sysfs on /sys type sysfs (rw)
devpts on /dev/pts type devpts (rw,gid=5,mode=620)
tmpfs on /dev/shm type tmpfs (rw,rootcontext="system_u:object_r:tmpfs_t:s0")
/dev/vda1 on /boot type ext4 (rw)
none on /proc/sys/fs/binfmt_misc type binfmt_misc (rw)
/dev/mapper/GLUSTER_vg1-GLUSTER_lv1 on /gluster/brick1 type xfs (rw,noatime,nodiratime,inode64)
[root@rhgs-test-2 ~]# df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/mapper/vg_rhgstest2-lv_root
                 35G   2.4G   31G    8% /
tmpfs            1004M    0 1004M    0% /dev/shm
/dev/vda1        477M   36M  416M    8% /boot
/dev/mapper/GLUSTER_vg1-GLUSTER_lv1
                 50G   34M   50G    1% /gluster/brick1
[root@rhgs-test-2 ~]#
```

To see volume groups a.k.a vgs, run the following command:

```
$ vgs
```

```
[root@rhgs-test-2 ~]# vgs
VG                #PV #LV #SN Attr   VSize  VFree
GLUSTER_vg1       1   2   0 wz--n- 50.00g    0
vg_rhgstest2      1   2   0 wz--n- 39.51g    0
```

To check physical volume, run the following command:

```
$ pvs
```

```
[root@rhgs-test-2 ~]# pvs
PV                VG                Fmt  Attr PSize  PFree
/dev/vda2         vg_rhgstest2  lvm2 a--  39.51g    0
/dev/vdb          GLUSTER_vg1  lvm2 a--  50.00g    0
```

And to check logical volume, run the following command:

```
$ lvs
```

```
[root@rhgs-test-2 ~]# lvs
LV                VG                Attr      LSize  Pool              Origin Data%  Meta%  Move Log Cpy%Sync Convert
GLUSTER_lv1       GLUSTER_vg1       Vwi-aot--- 49.75g GLUSTER_pool1
GLUSTER_pool1     GLUSTER_vg1       twi-aot--- 49.75g
lv_root           vg_rhgstest2      -wi-ao--- 35.57g
lv_swap           vg_rhgstest2      -wi-ao---  3.94g
```

We can see that volume groups and logical volumes has been successfully created.

2.12.4 How to start and stop services

‘service’ section lets you start, stop, enable, disable, restart, or reload services. Multiple service names can be provided as a comma separated list as shown in configuration file below.

For a hands-on walkthrough of the process, let us take a look at how we can start services such as glusterd, httpd, ntp through the config file.

Here, we’ll learn how to start services such as glusterd, httpd, ntp through config file.

Step 1:

Create an empty file and give it any arbitrary name. For the purpose of this demonstration, let’s call our file `starting_services.conf`. Add the following lines to your newly created config file:

```
# This is a mandatory section, and hostnames/ip-address are listed one per line.
# IP address for host shown here is for demonstration, don't forget to change it
# to a valid IP address in your network.

[hosts]
10.209.69.106

# To start services

[service]
action=start
```

```
service=glusterd,httpd

# To stop services
#
# [service]
# action=stop
# service=glusterd,httpd

# To disable services
#
# [service]
# action=disable
# service=glusterd,httpd

# To restart services
#
# [service]
# action=restart
# service=glusterd,httpd

# To reload services
#
# [service]
# action=reload
# service=glusterd,httpd
```

Step 2:

Invoke gdeploy and run the file using:

```
gdeploy -c starting_services.conf
```

Step 3:

To check the status of glusterd service, run the following command:

```
$ systemctl status glusterd.service
```

```
[purnima@dhcp35-197 ~]$ systemctl status glusterd.service
● glusterd.service - GlusterFS, a clustered file-system server
   Loaded: loaded (/usr/lib/systemd/system/glusterd.service; disabled; vendor preset: disabled)
   Active: active (running) since Mon 2016-07-25 18:39:26 IST; 1min 55s ago
     Process: 25340 ExecStart=/usr/local/sbin/glusterd -p /var/run/glusterd.pid --log-level $LOG_LEVEL $GLUSTERD_OPTIONS (code=exited, status=0/SUCCESS)
    Main PID: 25347 (glusterd)
      CGroup: /system.slice/glusterd.service
              └─25347 /usr/local/sbin/glusterd -p /var/run/glusterd.pid --log-level INFO
```

To check the status of httpd service, run the following command:

```
$ systemctl status httpd.service
```

```
[purnima@dhcp35-197 ~]$ systemctl status httpd.service
● httpd.service - The Apache HTTP Server
   Loaded: loaded (/usr/lib/systemd/system/httpd.service; disabled; vendor preset: disabled)
   Active: active (running) since Mon 2016-07-25 18:39:33 IST; 2min 3s ago
     Main PID: 25552 (httpd)
    Status: "Total requests: 0; Idle/Busy workers 100/0; Requests/sec: 0; Bytes served/sec: 0 B/sec"
      CGroup: /system.slice/httpd.service
              └─25552 /usr/sbin/httpd -DFOREGROUND
                  └─25555 /usr/sbin/httpd -DFOREGROUND
                      └─25556 /usr/sbin/httpd -DFOREGROUND
                          └─25558 /usr/sbin/httpd -DFOREGROUND
```

As we can see the `gluster` and `httpd` services has been started.

2.12.5 Set/unset volume options on an existing volume

Often, we are required to customize our volume for different use cases. Setting those options enhances performance and prepares the volume for our desired task. In order to do so, we set different options on the volume using the ‘key’ parameter in our configuration file.

This tutorial will take you through just that. It’s intended to show you how you can set different volume options on an existing Gluster volume. (To create a Gluster volume, please refer to [1x3-gluster-volume](#) or [2x2-gluster-volume](#)).

2.12.6 Setting the options on an existing volume

Step 1:

Create an empty file and give it any arbitrary name. For the purpose of this demonstration, let’s call our file `set_options_vol.conf`. Add the following lines to your newly created config file:

```
# The config file sets configuration options for the already existing volume.

# A volume can be created and its volume options can be set at the time of creation.

# The volume section takes key value pairs. The number of keys should match
# the number of values.

# 'action' option specifies what action id to be performed in the volume.
# The choices are: [create, delete, add-brick, remove-brick, rebalance,
# set].

[volume]
action=set
volname=10.70.42.190:sample_volume
key=cluster.nufa,performance.cache-size,cluster.data-self-heal-algorithm
value=on,256MB,full
```

Step 2:

Invoke `gdeploy` and run the file using:

```
$ gdeploy -c set_options_vol.conf
```

```

Activities Terminator Tue 19:03
poo@poo:~/config
poo@poo:~/config poo@poo:~/config root@rhgs-test-2:~
poo@poo:~/config 100x31
[poo@poo config]$ vi set_options_vol.conf
[poo@poo config]$ vi set_options_vol.conf
[poo@poo config]$ gdeploy -c set_options_vol.conf

INFO: Volume management(action: set) triggered

PLAY [master] *****

TASK: [Sets options for volume] *****
changed: [10.70.42.190] => (item={'value': 'on', 'key': 'cluster.nufa'})
changed: [10.70.42.190] => (item={'value': '256MB', 'key': 'performance.cache-size'})
changed: [10.70.42.190] => (item={'value': 'full', 'key': 'cluster.data-self-heal-algorithm'})

PLAY RECAP *****
10.70.42.190 : ok=1 changed=1 unreachable=0 failed=0

[poo@poo config]$

```

Step 3:

To verify if these options are indeed set on the volume, run the following command:

```
$ gluster vol info
```

```

purnima@dhcp35-197:~ 122x29
[purnima@dhcp35-197 ~]$ ssh root@10.70.42.190
Last login: Sun Jul 24 18:17:41 2016 from vpn1-7-52.sin2.redhat.com
[root@rhgs-test-2 ~]# gluster vol info

Volume Name: sample_volume
Type: Distributed-Replicate
Volume ID: 37d7ea59-e408-4842-b9b7-fee440fc1de5
Status: Started
Number of Bricks: 2 x 2 = 4
Transport-type: tcp
Bricks:
Brick1: 10.70.42.190:/gluster/brick1/one
Brick2: 10.70.42.232:/gluster/brick1/one
Brick3: 10.70.43.127:/gluster/brick1/one
Brick4: 10.70.43.67:/gluster/brick1/one
Options Reconfigured:
cluster.data-self-heal-algorithm: full
performance.cache-size: 256MB
cluster.nufa: on
performance.readdir-ahead: on

```

We can see that those options have been set on the volume.

2.12.7 Resetting the options on the existing volume

What if you want to reset or unset some options that you no longer require? There is no `unset` option per se, but you can `set` them back to different values to unset them. This configuration will show you how to do it. Change the values in the value parameter and the rest of it will be taken care of by gdeploy based on the config file.

Step 1:

Let's call your reset configuration file `reset_options_vol.conf`. Add the following lines to `reset_options_vol.conf`:

```
# This config resets options for the volume

[volume]
action=set
volname=10.70.42.190:sample_volume
key=cluster.nufa,features.lock-heal
value=off,off
```

Step 2:

Invoke gdeploy and run the following command:

```
$ gdeploy -c reset_options_vol.conf
```

Step 3:

To verify if options have been reset, run the following command:

```
$ gluster vol info
```

You'll see that the desired settings have been applied on the volume.

2.12.8 Installing packages from yum repositories

`yum` section allows you to install or remove packages using `yum` package manager.

Note :

Make sure that your system is registered with subscription manager before trying to install packages otherwise you'll get an error while following the steps below.

Step 1:

Create an empty file and give it any arbitrary name. For the purpose of this demonstration, let's call our file `install_packages.conf`. Add the following lines to your newly created config file:

```
# To install package(s) :

# Make sure you have the appropriate values in all the placeholders shown in this_
↪configuration file.
# These values are just for demonstration purposes.

[yum]
action=install
repos=<reponames>
```

```
packages=vi,glusterfs
gpgcheck=no
update=no

# Explanation of the above parameters

# packages
# -----

# This takes a comma separate list of values that are packages names you
# wish to install.

# gpgcheck
# -----

# gpgcheck is set to `yes` by default. You can override it
# by setting it to `no` as illustrated above.

# update
# -----

# By default, gdeploy runs `yum update` before installation. To disable
# this behaviour, set update=no as shown above. The default value is `yes`.

# To remove package(s):
# [yum]
# action=remove
# packages=vi
```

Step 2:

As always, to invoke gdeploy run the following command:

```
$ gdeploy -c install_packages.conf
```

2.12.9 How to disable repos

To disable enabled repos, use the action 'disable-repos'. The required repos should be passed as value to repos option.

NOTE : If repos are not provided all the enabled repos will be disabled.

Step 1:

Create an empty file and give it any arbitrary name. For the purpose of this demonstration, let's call our file `disable.conf`. Add the following lines to your newly created config file:

```
[RH-subscription]
action=disable-repos
repos=fancy_repo1,fancy_repo2
```

Step 2:

Invoke gdeploy and run the following command:

```
$ gdeploy -c disable.conf
```

2.12.10 Quota setup on an existing volume

Here, we will be setting up quota on an existing volume.

Step 1:

Create an empty '.conf' file e.g. 'quota.conf' and add the following to it:

```
#
# Usage:
#     gdeploy -c quota.conf
#
# This config enables and sets up quota limit for the specified volume
#

[quota]
action=enable
volname=10.70.41.236:1x2_vol

#You can skip the above if quota is already enabled on the volume

[quota]
action=limit-usage
volname=10.70.41.236:1x2_vol
path=/
size=20MB
```

'1x2_vol' is the name of our volume and 10.70.41.236 is one of the hosts / nodes in the cluster.

Step 2:

Run this file using the following command:

```
$gdeploy -c quota.conf
```

```
[root@dhcp35-167 anubha]# vi quota.conf
[root@dhcp35-167 anubha]# gdeploy -c quota.conf

PLAY [master] *****
*****

TASK [Gluster volume quota limit size operation] *****
*****
[DEPRECATION WARNING]: Using bare variables is deprecated. Update
your
playbooks so that the environment value uses the full variable sy
ntax
('{{limits}}').
This feature will be removed in a future release. Deprecation
warnings can be disabled by setting deprecation_warnings=False in
ansible.cfg.
changed: [10.70.41.236] => (item={u'path': u'/', u'size': u'20MB'})

PLAY RECAP *****
10.70.41.236 : ok=1    changed=1    unreachable=0    failed=0

[root@dhcp35-167 anubha]#
```

Step 3:

You can check whether quota has been set using the command:

```
$gluster vol quota 1x2_vol list
```

This command should be run on the machine where the volume exists. ‘1x2_vol’ is the name of our volume.

```
root@gluster-anubha:~  
[anubha@dhcp35-167 ~]$ ssh root@10.70.41.236  
Last login: Fri Jul 22 18:35:27 2016 from vpn1-7-75.  
sin2.redhat.com  
[root@gluster-anubha ~]# gluster vol quota 1x2_vol list  
? Hard-limit exceeded?  
-----  
Path                Hard-limit  Soft-limit  Used  Available  Soft-limit exceeded  
-----  
/anu                 50Bytes    80%(40Bytes) 0Bytes 50Bytes      No  
/sample_directory   10.0MB     80%(8.0MB) 0Bytes 10.0MB      No  
/                    20.0MB     80%(16.0MB) 0Bytes 20.0MB      No  
[root@gluster-anubha ~]#
```

2.12.11 Enabling and disabling quota

This document is intended to demonstrate how one can enable and disable quota on a Gluster volume.

Step 1:

Create a ‘.conf’ file with the following contents:

```
[quota]  
action=enable  
volname=10.70.41.236:1x2_vol
```

Here, ‘1x2_vol’ is the name of our volume and 10.70.41.236 is one of the hosts / nodes in the cluster.

We’ll name this file ‘enable_quota.conf’.

Step 2:

Run this using:

```
gdeploy -c enable_quota.conf
```

```

Activities Terminator
Fri 19:34
anubha@dhcp35-167:/home/anubha
anubha@dhcp35-167:/home/anubha 112x31
[root@dhcp35-167 anubha]# gdeploy -c enable_quota.conf

PLAY [master] *****

TASK [Enabling quota for the volume] *****
changed: [10.70.41.236]

PLAY RECAP *****
10.70.41.236 : ok=1 changed=1 unreachable=0 failed=0

[root@dhcp35-167 anubha]#

```

Step 3:

You can check whether quota has been enabled by checking volume info using:

```
$gluster vol info
```

This command can be run on any of the machines on which the volume resides.

```

Activities Terminator
Fri 19:34
root@gluster-anubha:~
root@gluster-anubha ~ 112x31
[anubha@dhcp35-167 ~]$ ssh root@10.70.41.236
Last login: Fri Jul 22 19:33:55 2016 from vpn1-7-75.sin2.redhat.com
[root@gluster-anubha ~]# gluster vol info

Volume Name: 1x2_vol
Type: Replicate
Volume ID: dabd5906-63d4-40d7-b944-193c496cec68
Status: Started
Number of Bricks: 1 x 2 = 2
Transport-type: tcp
Bricks:
Brick1: 10.70.41.236: /mnt/data/1
Brick2: 10.70.42.253: /mnt/data/1
Options Reconfigured:
features.quota-deemstats: on
features.inode-quota: on
features.quota: on
performance.readdir-ahead: on
[root@gluster-anubha ~]#

```

We can see that quota has been enabled on this volume.

One may follow the following steps to disable quota on this volume.

Step 1:

Create a '.conf' file with the following contents:

```
[quota]
action=disable
volname=10.70.41.236:1x2_vol
```

Here, '1x2_vol' is the name of our volume and 10.70.41.236 is one of the hosts / nodes in the cluster.

We'll name this file 'disable_quota.conf'.

Step 2:

Run this using:

```
gdeploy -c disable_quota.conf
```

A screenshot of a terminal window titled 'anubha@dhcp35-167:~' with a window icon 'x' in the top right. The terminal shows the command 'gdeploy -c disable_quota.conf' being executed. The output includes a 'PLAY [master]' line, a 'TASK [Disabling quota for volume]' line, and a 'changed: [10.70.41.236]' line. Below this is a 'PLAY RECAP' section showing '10.70.41.236 : ok=1 changed=1 unreachable=0 failed=0'. The prompt '[anubha@dhcp35-167 ~]\$' is visible at the bottom.

```
Activities Terminator Fri 19:36
anubha@dhcp35-167:~
anubha@dhcp35-167:~ 11x31
[anubha@dhcp35-167 ~]$ gdeploy -c disable_quota.conf
PLAY [master] *****
TASK [Disabling quota for volume] *****
changed: [10.70.41.236]
PLAY RECAP *****
10.70.41.236 : ok=1 changed=1 unreachable=0 failed=0
[anubha@dhcp35-167 ~]$
```

Step 3:

You can check whether quota has been disabled by checking volume info using:

```
$gluster vol info
```

This command can be run on any of the machines on which the volume resides.

```

Activities Terminator
Fri 19:36
root@gluster-anubha:~
root@gluster-anubha ~ 112x31
[anubha@dhep35-167 ~]$ ssh root@10.70.41.236
Last login: Fri Jul 22 19:36:09 2016 from vpn1-7-75.sin2.redhat.com
[root@gluster-anubha ~]# gluster vol info

Volume Name: 1x2_vol
Type: Replicate
Volume ID: dabd5906-63d4-40d7-b944-193c496cec68
Status: Started
Number of Bricks: 1 x 2 = 2
Transport-type: tcp
Bricks:
Brick1: 10.70.41.236: /mnt/data/1
Brick2: 10.70.42.253: /mnt/data/1
Options Reconfigured:
features.inode-quota: off
features.quota: off
performance.readdir-ahead: on
[root@gluster-anubha ~]#

```

2.12.12 Create a Gluster volume and set up quota

Here, we will see how we can create a 1x2 replica volume and then set a size limit on one of the directories within it. A 1x2 replica volume means that we would need 2 bricks and each file will have 2 replicas, one on each brick.

As a recommended practice, our bricks reside on separate machines. We have used two VMs for our two bricks in our case, and these have IPs 10.70.41.236 and 10.70.42.253.

Step 1:

Create an empty '.conf' file e.g. '1x2volume.conf' on the machine where you have gdeploy installed and add the following lines to it:

```

# This does backend setup first and then creates the volume using the
# setup bricks.

[hosts]
10.70.41.236
10.70.42.253

# Common backend setup for 2 of the hosts.
[backend-setup]
devices=vda
mountpoints=/mnt/data
brick_dirs=/mnt/data/1

# If backend-setup is different for each host
# [backend-setup:192.168.122.109]
# devices=sdb
# brick_dirs=/gluster/brick/brick1
#
# [backend-setup:192.168.122.227]

```

```
# devices=sda,sdb,sdc
# brick_dirs=/gluster/brick/brick{1,2,3}
#
[peer]
manage=probe

[volume]
action=create
volname=1x2_vol
replica=yes
replica_count=2
force=yes

[clients]
action=mount
volname=1x2_vol
hosts=10.70.41.236
fstype=glusterfs
client_mount_points=/glusterfs

#Enabling quota for this volume
[quota]
action=enable
volname=10.70.41.236:1x2_vol

# This will set up a quota limit for the specified path on the volume
[quota]
action=limit-usage
volname=10.70.41.236:1x2_vol
path=/sample_directory
size=10MB
```

path refers to a directory on the client mount point i.e. a directory inside /glusterfs in our case. We created a directory named “sample_directory” inside “/glusterfs” on our client machine 10.70.41.236 using the “mkdir” command. We are setting a size limit on this directory.

Step 2:

Invoke gdeploy and run the file using:

```
$gdeploy -c 1x2volume.conf
```

‘1x2volume.conf’ is the name of our configuration file.

```

anubha@dhcp35-167:~
File Edit View Search Terminal Help
[anubha@dhcp35-167 ~]$ ls
lx2volume.conf  Diagram1.dia~  fscreate.retry  mountlx2vol  Pictures  Videos
Desktop         Documents      gdeploy         mount_host   Public    work
Diagram1.dia    Downloads      gitwebsite      Music        Templates
[anubha@dhcp35-167 ~]$ gdeploy -c lx2volume.conf

PLAY [gluster_servers] *****

TASK [Create Physical Volume] *****
changed: [10.70.41.236] => (item=/dev/vda)
changed: [10.70.42.253] => (item=/dev/vda)

PLAY RECAP *****
10.70.41.236      : ok=1  changed=1  unreachable=0  failed=0
10.70.42.253      : ok=1  changed=1  unreachable=0  failed=0

PLAY [gluster_servers] *****

TASK [Create volume group on the disks] *****
changed: [10.70.42.253] => (item={u'brick': u'/dev/vda', u'vg': u'GLUSTER_vg1'})
changed: [10.70.41.236] => (item={u'brick': u'/dev/vda', u'vg': u'GLUSTER_vg1'})

PLAY RECAP *****
10.70.41.236      : ok=1  changed=1  unreachable=0  failed=0
10.70.42.253      : ok=1  changed=1  unreachable=0  failed=0

PLAY [gluster_servers] *****

TASK [Create logical volume named metadata] *****
changed: [10.70.41.236] => (item=GLUSTER_vg1)
changed: [10.70.42.253] => (item=GLUSTER_vg1)

TASK [create data LV that has a size which is a multiple of stripe width] *****
changed: [10.70.41.236] => (item={u'lv': u'GLUSTER_lv1', u'pool': u'GLUSTER_pool1', u'vg': u'GLUSTER_vg1'})
changed: [10.70.42.253] => (item={u'lv': u'GLUSTER_lv1', u'pool': u'GLUSTER_pool1', u'vg': u'GLUSTER_vg1'})

```

```

anubha@dhcp35-167:~
File Edit View Search Terminal Help
PLAY [gluster_servers] *****

TASK [Create logical volume named metadata] *****
changed: [10.70.41.236] => (item=GLUSTER_vg1)
changed: [10.70.42.253] => (item=GLUSTER_vg1)

TASK [create data LV that has a size which is a multiple of stripe width] *****
changed: [10.70.41.236] => (item={u'lv': u'GLUSTER_lv1', u'pool': u'GLUSTER_pool1', u'vg': u'GLUSTER_vg1'})
changed: [10.70.42.253] => (item={u'lv': u'GLUSTER_lv1', u'pool': u'GLUSTER_pool1', u'vg': u'GLUSTER_vg1'})

TASK [Convert the logical volume] *****
changed: [10.70.41.236] => (item={u'lv': u'GLUSTER_lv1', u'pool': u'GLUSTER_pool1', u'vg': u'GLUSTER_vg1'})
changed: [10.70.42.253] => (item={u'lv': u'GLUSTER_lv1', u'pool': u'GLUSTER_pool1', u'vg': u'GLUSTER_vg1'})

TASK [create stripe-aligned thin volume] *****
changed: [10.70.41.236] => (item={u'lv': u'GLUSTER_lv1', u'pool': u'GLUSTER_pool1', u'vg': u'GLUSTER_vg1'})
changed: [10.70.42.253] => (item={u'lv': u'GLUSTER_lv1', u'pool': u'GLUSTER_pool1', u'vg': u'GLUSTER_vg1'})

TASK [Change the attributes of the logical volume] *****
changed: [10.70.42.253] => (item={u'lv': u'GLUSTER_lv1', u'pool': u'GLUSTER_pool1', u'vg': u'GLUSTER_vg1'})
changed: [10.70.41.236] => (item={u'lv': u'GLUSTER_lv1', u'pool': u'GLUSTER_pool1', u'vg': u'GLUSTER_vg1'})

PLAY RECAP *****
10.70.41.236      : ok=5  changed=5  unreachable=0  failed=0
10.70.42.253      : ok=5  changed=5  unreachable=0  failed=0

PLAY [gluster_servers] *****

TASK [Create a xfs filesystem] *****
changed: [10.70.42.253] => (item=/dev/GLUSTER_vg1/GLUSTER_lv1)
changed: [10.70.41.236] => (item=/dev/GLUSTER_vg1/GLUSTER_lv1)

PLAY RECAP *****
10.70.41.236      : ok=1  changed=1  unreachable=0  failed=0
10.70.42.253      : ok=1  changed=1  unreachable=0  failed=0

```

```

anubha@dhcp35-167:~
File Edit View Search Terminal Help
10.70.42.253 : ok=1 changed=1 unreachable=0 failed=0

PLAY [gluster_servers] *****

TASK [Create the backend disks, skips if present] *****
changed: [10.70.42.253] => (item={u'device': u'/dev/GLUSTER_vg1/GLUSTER_lv1', u'path': u'/mnt/data'})
ok: [10.70.41.236] => (item={u'device': u'/dev/GLUSTER_vg1/GLUSTER_lv1', u'path': u'/mnt/data'})

TASK [Mount the volumes] *****
changed: [10.70.41.236] => (item={u'device': u'/dev/GLUSTER_vg1/GLUSTER_lv1', u'path': u'/mnt/data'})
changed: [10.70.42.253] => (item={u'device': u'/dev/GLUSTER_vg1/GLUSTER_lv1', u'path': u'/mnt/data'})

PLAY RECAP *****
10.70.41.236 : ok=2 changed=1 unreachable=0 failed=0
10.70.42.253 : ok=2 changed=2 unreachable=0 failed=0

Warning: We could not find the operations corresponding to the action specified for the section peer. Skipping this section.

PLAY [gluster_servers] *****

TASK [start/stop/restart/reload services] *****
ok: [10.70.42.253] => (item=glusterd)
ok: [10.70.41.236] => (item=glusterd)

PLAY RECAP *****
10.70.41.236 : ok=1 changed=0 unreachable=0 failed=0
10.70.42.253 : ok=1 changed=0 unreachable=0 failed=0

PLAY [gluster_servers] *****

TASK [Create the brick dirs, skips if present] *****
changed: [10.70.41.236] => (item=/mnt/data/l)
changed: [10.70.42.253] => (item=/mnt/data/l)

```

```

anubha@dhcp35-167:~
File Edit View Search Terminal Help

PLAY RECAP *****
10.70.41.236 : ok=1 changed=1 unreachable=0 failed=0
10.70.42.253 : ok=1 changed=1 unreachable=0 failed=0

PLAY [master] *****

TASK [Creates a Trusted Storage Pool] *****
changed: [10.70.41.236]

TASK [Pause for some seconds] *****
Pausing for 5 seconds
(ctrl+C then 'C' = continue early, ctrl+C then 'A' = abort)
ok: [10.70.41.236]

PLAY RECAP *****
10.70.41.236 : ok=2 changed=1 unreachable=0 failed=0

PLAY [master] *****

TASK [Creates a volume] *****
changed: [10.70.41.236]

PLAY RECAP *****
10.70.41.236 : ok=1 changed=1 unreachable=0 failed=0

PLAY [master] *****

TASK [Starts a volume] *****
changed: [10.70.41.236]

PLAY RECAP *****
10.70.41.236 : ok=1 changed=1 unreachable=0 failed=0

```

```

anubha@dhcp35-167:~$
File Edit View Search Terminal Help

PLAY [clients] *****

TASK [Create the dir to mount the volume, skips if present] *****
changed: [10.70.41.236] => (item={u'mountpoint': u'/glusterfs', u'fstype': u'fuse'})

PLAY RECAP *****
10.70.41.236      : ok=1    changed=1    unreachable=0    failed=0

PLAY [clients] *****

TASK [Mount the volumes, if fstype is glusterfs] *****
changed: [10.70.41.236] => (item={u'mountpoint': u'/glusterfs', u'fstype': u'fuse'})

PLAY RECAP *****
10.70.41.236      : ok=1    changed=1    unreachable=0    failed=0

PLAY [clients] *****

TASK [Mount the volumes if fstype is NFS] *****
skipping: [10.70.41.236] => (item={u'mountpoint': u'/glusterfs', u'fstype': u'fuse'})

PLAY RECAP *****
10.70.41.236      : ok=0    changed=0    unreachable=0    failed=0

PLAY [clients] *****

TASK [Mount the volumes, if fstype is CIFS] *****
skipping: [10.70.41.236] => (item={u'mountpoint': u'/glusterfs', u'fstype': u'fuse'})

PLAY RECAP *****
10.70.41.236      : ok=0    changed=0    unreachable=0    failed=0

[anubha@dhcp35-167 ~]$

```

```

anubha@dhcp35-167:/home/anubha
anubha@dhcp35-167:/home/anubha 11x31

PLAY RECAP *****
10.70.41.236      : ok=0    changed=0    unreachable=0    failed=0

PLAY [clients] *****

TASK [Mount the volumes, if fstype is CIFS] *****
[DEPRECATION WARNING]: Using bare variables is deprecated. Update your playbooks so that the environment value
uses the full variable syntax ('{{client_mounts}}').
This feature will be removed in a future release. Deprecation
warnings can be disabled by setting deprecation_warnings=False in ansible.cfg.
skipping: [10.70.41.236] => (item={u'mountpoint': u'/glusterfs', u'fstype': u'fuse'})

PLAY RECAP *****
10.70.41.236      : ok=0    changed=0    unreachable=0    failed=0

PLAY [master] *****

TASK [Gluster volume quota limit size operation] *****
[DEPRECATION WARNING]: Using bare variables is deprecated. Update your playbooks so that the environment value
uses the full variable syntax ('{{limits}}').
This feature will be removed in a future release. Deprecation
warnings can be disabled by setting deprecation_warnings=False in ansible.cfg.
changed: [10.70.41.236] => (item={u'path': u'/sample_directory', u'size': u'10MB'})

PLAY RECAP *****
10.70.41.236      : ok=1    changed=1    unreachable=0    failed=0

[root@dhcp35-167 anubha]#

```

Step 3:

You can check whether a gluster volume is created by running the following command on any or all of the nodes:

```
$gluster vol info
```

```

root@gluster-anubha:~
[anubha@dhep35-167 ~]$ ssh root@10.70.41.236
Last login: Fri Jul 22 17:59:04 2016 from vpn1-7-75.sin2.redhat.com
[root@gluster-anubha ~]# gluster vol info

Volume Name: 1x2_vol
Type: Replicate
Volume ID: dabd5906-63d4-40d7-b944-193c496cec68
Status: Started
Number of Bricks: 1 x 2 = 2
Transport-type: tcp
Bricks:
Brick1: 10.70.41.236: /mnt/data/1
Brick2: 10.70.42.253: /mnt/data/1
Options Reconfigured:
features.quota-deemstatfs: on
features.inode-quota: on
features.quota: on
performance.readdir-ahead: on
[root@gluster-anubha ~]#

```

Here, we can also see that quota has been enabled.

Step 4:

Let's check whether the size limit for our directory "sample_directory" has been set. One can check quota attributes on a volume using the command:

```
$gluster vol quota 1x2_vol list
```

Here, 1x2_vol is the name of our volume.

```

root@gluster-anubha:~
[root@gluster-anubha ~]# gluster vol quota 1x2_vol list
Path                               Hard-limit  Soft-limit  Used  Available  Soft-limit exceeded?  Hard-
limit exceeded?
-----
/anu                               50Bytes     80%(40Bytes)  0Bytes  50Bytes                No
/sample_directory                 10.0MB      80%(8.0MB)   0Bytes  10.0MB                No
[root@gluster-anubha ~]#

```

Step 5:

You can test the volume by creating a file and see whether it is getting replicated. On your client machine (10.70.41.236 in our case), traverse to the path you have mentioned under “client_mount_points” (e.g. ‘cd /glusterfs’) and create a file using the following command:

```
$touch sample.txt
```

This command will create a file named as “sample.txt” under the directory “/glusterfs”. You may create this file on any of the directories under “/glusterfs”, we have created it in the topmost one.

You can check whether the file has been replicated twice by traversing to the path “/mnt/data1/1” on both the nodes and running the command:

```
$ls
```

You will see two copies of your file in total, on the bricks.

You have successfully setup a 1x2 Gluster volume using gdeploy and set a size limit on one of the directories on it.

2.12.13 Set a 5GB limit on a directory using quota

Here, we will see how to set a 5GB limit on a directory within our volume using quota.

Step 1:

Create the following ‘.conf’ file:

```
#Enabling quota for this volume
[quota]
action=enable
volname=10.70.41.236:1x2_vol

# This will set up a quota limit for the specified path on the volume
[quota]
action=limit-usage
volname=10.70.41.236:1x2_vol
path=/main_dir
size=5GB
```

Step 2:

Run the file using:

```
$gdeploy -c 5gbquota.conf
```

Here, ‘5gbquota.conf’ is the name of our configuration file created in Step 1.

```

anubha@dhcp35-167:/home/anubha
anubha@dhcp35-167:/home/anubha 112x29
root@dhcp35-167 anubha]# gdeploy -c 5gbquota.conf

PLAY [master] *****

TASK [Gluster volume quota limit size operation] *****
[DEPRECATION WARNING]: Using bare variables is deprecated. Update your playbooks so that the environment value
uses the full variable syntax ('{{limits}}').
This feature will be removed in a future release. Deprecation
warnings can be disabled by setting deprecation_warnings=False in ansible.cfg.
changed: [10.70.41.236] => (item={u'path': u'/main_dir', u'size': u'5GB'})

PLAY RECAP *****
10.70.41.236      : ok=1    changed=1    unreachable=0    failed=0

root@dhcp35-167 anubha]#

```

Step 3:

You can check whether quota is enabled on your desired volume by checking volume information:

```
$gluster vol info
```

This command needs to be run on the any or all of the machines on which the volume resides.

Step 4:

To check whether the size limit of 5GB has been set, we run the command:

```
$gluster vol quota 1x2_vol list
```

This command gives us a detailed description of quota settings applied on our volume.

```

root@gluster-anubha glusterfs]# gluster volume quota 1x2_vol list
Path                Hard-limit  Soft-limit  Used  Available  Soft-limit exceeded?  Hard-
-----
main_dir            5.0GB      80%(4.0GB)  0Bytes  5.0GB              No
root@gluster-anubha glusterfs]#

```

2.12.14 Creating a volume and setting a tuning profile on it

This document will walk you through how you can create a Gluster volume and set a profile on it. Profiles are directories of files that contain settings to enhance performance of a volume. There are many profiles that come with Red Hat Gluster Storage and these are tailored for different workloads. One can also define or create a new profile. As profiles aid in performance tuning (improving system performance), they are also called as “tuning profiles”.

Pre-defined profiles can be found here as subdirectories: /etc/tune-profiles.

For instance, /etc/tune-profiles/virtual-guest contains all the files and settings for the virtual-guest profile, which is a profile that sets performance options for virtual machines.

The following steps will illustrate how to create a volume and set a tuning profile on it.

Step 1:

Create the following configuration file:

```
[hosts]
10.70.41.236
10.70.42.253

# Common backend setup for 2 of the hosts.
[backend-setup]
devices=vda
mountpoints=/mnt/data
brick_dirs=/mnt/data/1

# If backend-setup is different for each host
# [backend-setup:192.168.122.109]
# devices=sdb
# brick_dirs=/gluster/brick/brick1
#
# [backend-setup:192.168.122.227]
# devices=sda,sdb,sdc
# brick_dirs=/gluster/brick/brick{1,2,3}
#
[peer]
manage=probe

[volume]
action=create
volname=1x2_vol
replica=yes
replica_count=2
force=yes

[clients]
action=mount
volname=1x2_vol
hosts=10.70.41.236
fstype=glusterfs
client_mount_points=/glusterfs

#The above section creates the volume. The below section will apply a profile to it.

[tune-profile]
rhgs-sequential-io

#This will set the profile 'rhgs-sequential-io'.
```

Step 2:

Invoke gdeploy and run this using:

```
$gdeploy -c tune_profile.conf
```

where “tune_profile.conf” is the name of our configuration file created in Step 1.

Step 3:

Check whether this has been applied using:

```
$tuned-adm list
```

This command, when run on any of the hosts / cluster nodes, will return you the list of available profiles along with the current active profile. In our case, the current active profile would be ‘rhgs-sequential-io’.

2.12.15 Setting a tuning profile on an existing volume

2.12.16 NFS Ganesha setup end-to-end

Here we’ll see how we can setup NFS Ganesha using gdeploy. We’ll be writing a configuration file for the end-to-end setup, right from creating a volume, subscribing to channels and installing the right packages. This configuration file will also create a high availability cluster and export the volume.

Step 1:

Create an empty .conf file with the following:

```
[hosts]
dhcp37-102.lab.eng.blr.redhat.com
dhcp37-103.lab.eng.blr.redhat.com

[backend-setup]
devices=/dev/vdb
vgs=vgl
pools=pool1
lvs=lv1
mountpoints=/mnt/brick

# Subscribe to necessary channels
[RH-subscription1]
action=register
username=<username>
password=<password>
pool=<pool>

[RH-subscription2]
action=disable-repos
repos=

[RH-subscription3]
action=enable-repos
repos=rhel-7-server-rpms,rh-gluster-3-for-rhel-7-server-rpms,rh-gluster-3-nfs-for-
↪rhel-7-server-rpms,rhel-ha-for-rhel-7-server-rpms

#Installing nfs-ganesha
[yum]
```

```

action=install
repolist=
pgpcheck=no
update=no
packages=glusterfs-ganesha

#Enabling the firewall service and configuring necessary ports
[firewalld]
action=add
ports=111/tcp,2049/tcp,54321/tcp,5900/tcp,5900-6923/tcp,5666/tcp,16514/tcp,662/tcp,
↪662/udp
services=glusterfs,nlm,nfs,rpc-bind,high-availability,mountd,rquota

#This will create a volume. Skip this section if your volume already exists
[volume]
action=create
volname=ganesha
transport=tcp
replica_count=2
force=yes

#Creating a high availability cluster and exporting the volume
[nfs-ganesha]
action=create-cluster
ha-name=ganesha-ha-360
cluster-nodes=dhcp37-102.lab.eng.blr.redhat.com,dhcp37-103.lab.eng.blr.redhat.com
vip=10.70.44.121,10.70.44.122
volname=ganesha

```

Step 2:

Run this file using:

```
$gdeploy -c nfs_ganeshal.conf
```

where `nfs_ganeshal.conf` is the name of our configuration file saved in Step 1.

Step 3:

To see if your volume has been exported, you may run this command on any or all of the nodes:

```
$showmount -e localhost
```

2.12.17 Unexporting a volume and destroying an NFS-Ganesha HA Cluster

Here, we'll see how we can unexport a volume and destroy a high availability cluster.

Step 1:

Create the following configuration file:

```

[hosts]
dhcp37-102.lab.eng.blr.redhat.com
dhcp37-103.lab.eng.blr.redhat.com

# To un-export the volume:

[nfs-ganeshal]

```

```
action=unexport-volume
volname=ganeshha

# To destroy the high availability cluster

[nfs-ganeshha2]
action=destroy-cluster
cluster-nodes=dhcp37-102.lab.eng.blr.redhat.com,dhcp37-103.lab.eng.blr.redhat.com
```

‘ganeshha’ is the name of our volume.

Step 2:

Run this file using:

```
$gdeploy -c ganeshha_destroy.conf
```

Here, “ganeshha_destroy.conf” is the name of our configuration file created in Step 1.

Step 3:

Now, when you run this command on any or all of the nodes in the cluster, you will not see any mounts for nfs-ganeshha:

```
$showmount -e localhost
```

You have successfully unexported the volume and destroyed the HA cluster.

CHAPTER 3

Indices and tables

- `genindex`
- `modindex`
- `search`