



Red Hat OpenShift Container Storage 3.10

Operations Guide

Configuring and Managing Red Hat OpenShift Container Storage.

Red Hat Openshift Container Storage3.10 Operations Guide

Configuring and Managing Red Hat Openshift Container Storage.

Bhavana Mohan
Customer Content Services Red Hat
bmohanra@redhat.com

Legal Notice

Copyright © 2018 Red Hat, Inc.

This document is licensed by Red Hat under the [Creative Commons Attribution-ShareAlike 3.0 Unported License](#). If you distribute this document, or a modified version of it, you must provide attribution to Red Hat, Inc. and provide a link to the original. If the document is modified, all Red Hat trademarks must be removed.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat Software Collections is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Abstract

This guide provides information about operating your Container Storage deployment.

Table of Contents

PART I. MANAGE	4
CHAPTER 1. MANAGING CLUSTERS	5
1.1. INCREASING STORAGE CAPACITY	5
1.2. REDUCING STORAGE CAPACITY	18
CHAPTER 2. OPERATIONS ON A RED HAT GLUSTER STORAGE POD IN AN OPENSIFT ENVIRONMENT	25
PART II. OPERATIONS	31
CHAPTER 3. CREATING PERSISTENT VOLUMES	32
3.1. FILE STORAGE	32
3.2. BLOCK STORAGE	53
CHAPTER 4. SHUTTING DOWN GLUSTER-BLOCK CLIENT NODES	69
CHAPTER 5. S3 COMPATIBLE OBJECT STORE IN A RED HAT OPENSIFT CONTAINER STORAGE ENVIRONMENT	70
5.1. SETTING UP S3 COMPATIBLE OBJECT STORE FOR RED HAT OPENSIFT CONTAINER STORAGE	70
5.2. OBJECT OPERATIONS	73
CHAPTER 6. CLUSTER ADMINISTRATOR SETUP	75
CHAPTER 7. GLUSTER BLOCK STORAGE AS BACKEND FOR LOGGING AND METRICS	76
7.1. PREREQUISITES	76
7.2. ENABLING GLUSTER BLOCK STORAGE AS BACKEND FOR LOGGING	76
7.3. ENABLING GLUSTER BLOCK STORAGE AS BACKEND FOR METRICS	77
7.4. VERIFYING IF GLUSTER BLOCK IS SETUP AS BACKEND	78
PART III. SECURITY	80
CHAPTER 8. ENABLING ENCRYPTION	81
8.1. PREREQUISITES	81
8.2. ENABLING ENCRYPTION FOR A NEW RED HAT OPENSIFT CONTAINER STORAGE SETUP	81
8.3. ENABLING ENCRYPTION FOR AN EXISTING RED HAT OPENSIFT CONTAINER STORAGE SETUP	84
8.4. DISABLING ENCRYPTION	86
PART IV. MIGRATION	89
CHAPTER 9. UPDATING THE REGISTRY WITH RED HAT OPENSIFT CONTAINER STORAGE AS THE STORAGE BACK-END	90
9.1. VALIDATING THE OPENSIFT CONTAINER PLATFORM REGISTRY DEPLOYMENT	90
9.2. CONVERTING THE OPENSIFT CONTAINER PLATFORM REGISTRY WITH RED HAT OPENSIFT CONTAINER STORAGE	92
PART V. MONITORING	97
CHAPTER 10. ENABLING VOLUME METRICS	98
10.1. ENABLING VOLUME METRICS FOR FILE STORAGE AND BLOCK STORAGE	98
PART VI. TROUBLESHOOT	101
CHAPTER 11. TROUBLESHOOTING	102

CHAPTER 12. CLIENT CONFIGURATION USING PORT FORWARDING 104

APPENDIX A. REVISION HISTORY 105

INDEX 106

PART I. MANAGE

CHAPTER 1. MANAGING CLUSTERS

Heketi allows administrators to add and remove storage capacity by managing either a single or multiple Red Hat Gluster Storage clusters.

1.1. INCREASING STORAGE CAPACITY

You can increase the storage capacity using any of the following ways:

- Adding devices
- Increasing cluster size
- Adding an entirely new cluster.

1.1.1. Adding New Devices

You can add more devices to existing nodes to increase storage capacity. When adding more devices, you must ensure to add devices as a set. For example, when expanding a distributed replicated volume with a replica count of replica 2, then one device should be added to at least two nodes. If using replica 3, then at least one device should be added to at least three nodes.

You can add a device either using CLI, or the API, or by updating the topology JSON file. The sections ahead describe using heketi CLI and updating topology JSON file. For information on adding new devices using API, see Heketi API https://github.com/heketi/heketi/wiki/API#device_add

1.1.1.1. Using Heketi CLI

Register the specified device. The following example command shows how to add a device `/dev/sde` to node `d6f2c22f2757bf67b1486d868dcb7794`:

```
# heketi-cli device add --name=/dev/sde --
node=d6f2c22f2757bf67b1486d868dcb7794
OUTPUT:
Device added successfully
```

1.1.1.2. Updating Topology File

You can add the new device to the node description in your topology JSON file which is used to setup the cluster. Then re-run the command to load the topology.

Following is an example where a new `/dev/sde` drive added to the node:

In the file:

```
{
    "node": {
        "hostnames": {
            "manage": [
                "node4.example.com"
            ],
            "storage": [
                "192.168.10.100"
            ]
        }
    }
}
```

```

        },
        "zone": 1
    },
    "devices": [
        "/dev/sdb",
        "/dev/sdc",
        "/dev/sdd",
        "/dev/sde"
    ]
}

```

Load the topology file:

```

# heketi-cli topology load --json=topology-sample.json
Found node 192.168.10.100 on cluster d6f2c22f2757bf67b1486d868dcb7794
Found device /dev/sdb
Found device /dev/sdc
Found device /dev/sdd
Adding device /dev/sde ... OK
Found node 192.168.10.101 on cluster d6f2c22f2757bf67b1486d868dcb7794
Found device /dev/sdb
Found device /dev/sdc
Found device /dev/sdd
Found node 192.168.10.102 on cluster d6f2c22f2757bf67b1486d868dcb7794
Found device /dev/sdb
Found device /dev/sdc
Found device /dev/sdd
Found node 192.168.10.103 on cluster d6f2c22f2757bf67b1486d868dcb7794
Found device /dev/sdb
Found device /dev/sdc
Found device /dev/sdd

```

1.1.2. Increasing Cluster Size

Another way to add storage to Heketi, is to add new nodes to the cluster. Like adding devices, you can add a new node to an existing cluster by either using CLI or API. When you add a new node to the cluster, you must register new devices to that node.

This section describes how to use Heketi CLI to increase the cluster size. For more information on adding new devices using API, see Heketi API: https://github.com/heketi/heketi/wiki/API#node_add



NOTE

For adding a node to be successful, ensure the ports are opened for glusterd communication. For more information about the ports, see https://access.redhat.com/documentation/en-us/red_hat_gluster_storage/3.4/html/installation_guide/port_information

1.1.2.1. Adding a Node to OCP Cluster

1. Scaleup the OCP cluster to add the new node. For more information see, https://access.redhat.com/documentation/en-us/openshift_container_platform/3.11/html-single/configuring_clusters/#adding-cluster-hosts_adding-hosts-to-cluster

**NOTE**

If the new node is already part of OCP cluster then skip this step and proceed with Step 2.

2. Configure the firewall rules:

**NOTE**

For adding a node to be successful, ensure the ports are opened for glusterd communication. For more information about the ports, see https://access.redhat.com/documentation/en-us/red_hat_gluster_storage/3.4/html/installation_guide/port_information

1. Add the following rules /etc/sysconfig/iptables file of the newly added glusterfs node:

```
-A OS_FIREWALL_ALLOW -p tcp -m state --state NEW -m tcp --dport
24007 -j ACCEPT
-A OS_FIREWALL_ALLOW -p tcp -m state --state NEW -m tcp --dport
24008 -j ACCEPT
-A OS_FIREWALL_ALLOW -p tcp -m state --state NEW -m tcp --dport
2222 -j ACCEPT
-A OS_FIREWALL_ALLOW -p tcp -m state --state NEW -m multiport --
dports 49152:49664 -j ACCEPT
-A OS_FIREWALL_ALLOW -p tcp -m state --state NEW -m tcp --dport
24010 -j ACCEPT
-A OS_FIREWALL_ALLOW -p tcp -m state --state NEW -m tcp --dport
3260 -j ACCEPT
-A OS_FIREWALL_ALLOW -p tcp -m state --state NEW -m tcp --dport
111 -j ACCEPT
```

2. Reload/restart the iptables:

```
# systemctl restart iptables
```

3. Execute the following steps to add labels to the node where the RHGS Container will be deployed:

1. Verify that Red Hat Openshift Container Storage is deployed and working as expected in the existing project by executing the following command:

```
# oc get ds
```

For example:

```
# oc get ds
NAME                                DESIRED   CURRENT   READY   UP-TO-DATE
AVAILABLE   NODE SELECTOR   AGE
glusterfs-storage 3           3         3       3
3            glusterfs=storage-host 1d
```

2. Add the label for each node which is newly added, where the Red Hat Gluster Storage pods are to be added for the new cluster:

```
# oc label node <NODE_NAME> glusterfs=<node_label>
```

where,

- **NODE_NAME**: is the name of the newly created node.
- **node_label**: The name that is used in the existing daemonset. This is the value you get in the previous step when you execute **oc get ds**.

For example:

```
# oc label node 192.168.90.3 glusterfs=storage-host
node "192.168.90.3" labeled
```

3. Verify if the Red Hat Gluster Storage pods are running on the newly added node by executing the following command:

Observe additional Gluster Storage pods spawned on these new nodes

```
# oc get pods
```

For example:

```
# oc get pods
NAME                                READY    STATUS    RESTARTS   AGE
glusterfs-356cf                    1/1      Running   0           30d
glusterfs-fh4gm                    1/1      Running   0           30d
glusterfs-hg4tk                    1/1      Running   0           30d
glusterfs-v759z                    0/1      Running   0           1m
```

You should see additional Gluster Storage pods, in this example 4 gluster pods instead of just 3 as before. It will take 1-2 minutes for them to become healthy. (i.e. glusterfs-v759z 0/1 not healthy yet).

4. Verify if the Red Hat Gluster Storage pods are running

```
# oc get pods -o wide -l glusterfs=storage-pod
```

1.1.2.2. Using Heketi CLI

Following shows an example of how to add new node in **zone 1** to **597fceb5d6c876b899e48f599b988f54** cluster using the CLI:

```
# heketi-cli node add --zone=1 --cluster=597fceb5d6c876b899e48f599b988f54
--management-host-name=node4.example.com --storage-host-
name=192.168.10.104
```

OUTPUT:

Node information:

Id: 095d5f26b56dc6c64564a9bc17338cbf

State: online

Cluster Id: 597fceb5d6c876b899e48f599b988f54

```
Zone: 1
Management Hostname node4.example.com
Storage Hostname 192.168.10.104
```

The following example command shows how to register **/dev/sdb** and **/dev/sdc** devices for **095d5f26b56dc6c64564a9bc17338cbf** node:

```
# heketi-cli device add --name=/dev/sdb --
node=095d5f26b56dc6c64564a9bc17338cbf
OUTPUT:
Device added successfully

# heketi-cli device add --name=/dev/sdc --
node=095d5f26b56dc6c64564a9bc17338cbf
OUTPUT:
Device added successfully
```

1.1.2.3. Updating Topology File

You can expand a cluster by adding a new node to your topology JSON file. When adding the new node you must add this node information **after** the existing ones so that the Heketi CLI identifies on which cluster this new node should be part of.

Following shows an example of how to add a new node and devices:

```
{
  "node": {
    "hostnames": {
      "manage": [
        "node4.example.com"
      ],
      "storage": [
        "192.168.10.104"
      ]
    },
    "zone": 1
  },
  "devices": [
    "/dev/sdb",
    "/dev/sdc"
  ]
}
```

Load the topology file:

```
# heketi-cli topology load --json=topology-sample.json
Found node 192.168.10.100 on cluster d6f2c22f2757bf67b1486d868dcb7794
Found device /dev/sdb
Found device /dev/sdc
Found device /dev/sdd
Found device /dev/sde
Found node 192.168.10.101 on cluster d6f2c22f2757bf67b1486d868dcb7794
Found device /dev/sdb
Found device /dev/sdc
```

```

Found device /dev/sdd
Found node 192.168.10.102 on cluster d6f2c22f2757bf67b1486d868dcb7794
Found device /dev/sdb
Found device /dev/sdc
Found device /dev/sdd
Found node 192.168.10.103 on cluster d6f2c22f2757bf67b1486d868dcb7794
Found device /dev/sdb
Found device /dev/sdc
Found device /dev/sdd
Creating node node4.example.com ... ID:
ff3375aca6d98ed8a004787ab823e293
Adding device /dev/sdb ... OK
Adding device /dev/sdc ... OK

```

1.1.3. Adding a New Cluster

Storage capacity can also be increased by adding new clusters of Red Hat Gluster Storage. New clusters can be added in the following two ways based on the requirement:

- Adding a new cluster to the existing Red Hat Openshift Container Storage
- Adding another Red Hat Openshift Container Storage cluster in a new project

1.1.3.1. Adding a New Cluster to the Existing Red Hat Openshift Container Storage

To add a new cluster to the existing Red Hat Openshift Container Storage, execute the following commands:

1. Verify that Red Hat Openshift Container Storage is deployed and working as expected in the existing project by executing the following command:

```
# oc get ds
```

For example:

```

# oc get ds
NAME                                DESIRED   CURRENT   READY   UP-TO-DATE
AVAILABLE   NODE SELECTOR   AGE
glusterfs-storage      3         3         3       3         3
glusterfs=storage-host 1d

```

2. Verify if the Red Hat Gluster Storage pods are running by executing the following command:

Observe additional Gluster Storage pods spawned on these new nodes

```
# oc get pods
```

For example:

```

# oc get pods
NAME                READY   STATUS    RESTARTS   AGE
glusterfs-356cf    1/1     Running   0          30d

```

```
glusterfs-fh4gm    1/1      Running    0          30d
glusterfs-hg4tk    1/1      Running    0          30d
glusterfs-v759z    0/1      Running    0          1m
```

You should see additional Gluster Storage pods, in this example 4 gluster pods instead of just 3 as before. It will take 1-2 minutes for them to become healthy. (i.e. glusterfs-v759z 0/1 not healthy yet).

3. Add the label for each node, where the Red Hat Gluster Storage pods are to be added for the new cluster to start by executing the following command:

```
# oc label node <NODE_NAME> glusterfs=<node_label>
```

where,

- `NODE_NAME`: is the name of the newly created node
- `node_label`: The name that is used in the existing daemonset.

For example:

```
# oc label node 192.168.90.3 glusterfs=storage-host
node "192.168.90.3" labeled
```

4. Verify if the Red Hat Gluster Storage pods are running by executing the following command:

```
# oc get ds
```

For example:

```
# oc get ds
NAME                                DESIRED    CURRENT    READY    UP-TO-DATE
AVAILABLE    NODE SELECTOR    AGE
glusterfs-storage    3            3            3            3
glusterfs=storage-host    1d
```

5. Create a new topology file for the new cluster. You must provide a topology file for the new cluster which describes the topology of the Red Hat Gluster Storage nodes and their attached storage devices. As a sample, a formatted topology file (topology-sample.json) is installed with the 'heketi-client' package in the /usr/share/heketi/ directory.

For example:

```
{
  "clusters": [
    {
      "nodes": [
        {
          "node": {
            "hostnames": {
              "manage": [
                "node1.example.com"
              ],
              "storage": [
```

```

        "192.168.68.3"
      ],
    },
    "zone": 1
  },
  "devices": [
    "/dev/sdb",
    "/dev/sdc",
    "/dev/sdd",
    "/dev/sde",
    "/dev/sdf",
    "/dev/sdg",
    "/dev/sdh",
    "/dev/sdi"
  ]
},
{
  "node": {
    "hostnames": {
      "manage": [
        "node2.example.com"
      ],
      "storage": [
        "192.168.68.2"
      ]
    },
    "zone": 2
  },
  "devices": [
    "/dev/sdb",
    "/dev/sdc",
    "/dev/sdd",
    "/dev/sde",
    "/dev/sdf",
    "/dev/sdg",
    "/dev/sdh",
    "/dev/sdi"
  ]
},

```

```

.....
.....

```

where,

- clusters: Array of clusters.

Each element on the array is a map which describes the cluster as follows.

- nodes: Array of OpenShift nodes that will host the Red Hat Gluster Storage container

Each element on the array is a map which describes the node as follows

- node: It is a map of the following elements:
 - zone: The value represents the zone number that the node belongs to; the zone number is used by heketi for choosing optimum position of bricks by having

replicas of bricks in different zones. Hence zone number is similar to a failure domain.

- **hostnames:** It is a map which lists the manage and storage addresses
 - **manage:** It is the hostname/IP Address that is used by Heketi to communicate with the node
 - **storage:** It is the IP address that is used by other OpenShift nodes to communicate with the node. Storage data traffic will use the interface attached to this IP. This must be the IP address and not the hostname because, in an OpenShift environment, Heketi considers this to be the endpoint too.
- **devices:** Name of each disk to be added

Edit the topology file based on the Red Hat Gluster Storage pod hostname under the **node.hostnames.manage** section and **node.hostnames.storage** section with the IP address. For simplicity, the `/usr/share/heketi/topology-sample.json` file only sets up 4 nodes with 8 drives each.

6. For the existing cluster, `heketi-cli` will be available to load the new topology. Run the command to add the new topology to heketi:

```
# heketi-cli topology load --json=<topology file path>
```

For example:

```
# heketi-cli topology load --json=topology.json
Creating cluster ... ID: 94877b3f72b79273e87c1e94201ecd58
  Creating node node4.example.com ... ID:
  95cefa174c7210bd53072073c9c041a3
    Adding device /dev/sdb ... OK
    Adding device /dev/sdc ... OK
    Adding device /dev/sdd ... OK
    Adding device /dev/sde ... OK
  Creating node node5.example.com ... ID:
  f9920995e580f0fe56fa269d3f3f8428
    Adding device /dev/sdb ... OK
    Adding device /dev/sdc ... OK
    Adding device /dev/sdd ... OK
    Adding device /dev/sde ... OK
  Creating node node6.example.com ... ID:
  73fe4aa89ba35c51de4a51ecbf52544d
    Adding device /dev/sdb ... OK
    Adding device /dev/sdc ... OK
    Adding device /dev/sdd ... OK
    Adding device /dev/sde ... OK
```

1.1.3.2. Adding Another Red Hat Openshift Container Storage Cluster in a New Project

To add another Red Hat Openshift Container Storage in a new project to, execute the following commands:



NOTE

As Node label is global, there can be conflicts to start Red Hat Gluster Storage DaemonSets with the same label in two different projects. Node label is an argument to `cns-deploy`, thereby enabling deployment of multiple trusted storage pool by using a different label in a different project.

1. Create a new project by executing the following command:

```
# oc new-project <new_project_name>
```

For example:

```
# oc new-project storage-project-2
```

```
Now using project "storage-project-2" on server
"https://master.example.com:8443"
```

2. After the project is created, execute the following command on the master node to enable the deployment of the privileged containers as Red Hat Gluster Storage container can only run in the privileged mode.

```
# oc adm policy add-scc-to-user privileged -z storage-project-2
# oc adm policy add-scc-to-user privileged -z default
```

3. Create a new topology file for the new cluster. You must provide a topology file for the new cluster which describes the topology of the Red Hat Gluster Storage nodes and their attached storage devices. As a sample, a formatted topology file (`topology-sample.json`) is installed with the 'heketi-client' package in the `/usr/share/heketi/` directory.

For example:

```
{
  "clusters": [
    {
      "nodes": [
        {
          "node": {
            "hostnames": {
              "manage": [
                "node1.example.com"
              ],
              "storage": [
                "192.168.68.3"
              ]
            },
            "zone": 1
          },
          "devices": [
            "/dev/sdb",
            "/dev/sdc",
            "/dev/sdd",
            "/dev/sde",
            "/dev/sdf",

```

```

        "/dev/sdg",
        "/dev/sdh",
        "/dev/sdi"
    ],
},
{
    "node": {
        "hostnames": {
            "manage": [
                "node2.example.com"
            ],
            "storage": [
                "192.168.68.2"
            ]
        },
        "zone": 2
    },
    "devices": [
        "/dev/sdb",
        "/dev/sdc",
        "/dev/sdd",
        "/dev/sde",
        "/dev/sdf",
        "/dev/sdg",
        "/dev/sdh",
        "/dev/sdi"
    ]
},

```

.....

where,

- clusters: Array of clusters.

Each element on the array is a map which describes the cluster as follows.

- nodes: Array of OpenShift nodes that will host the Red Hat Gluster Storage container

Each element on the array is a map which describes the node as follows

- node: It is a map of the following elements:
 - zone: The value represents the zone number that the node belongs to; the zone number is used by heketi for choosing optimum position of bricks by having replicas of bricks in different zones. Hence zone number is similar to a failure domain.
 - hostnames: It is a map which lists the manage and storage addresses
 - manage: It is the hostname/IP Address that is used by Heketi to communicate with the node
 - storage: It is the IP address that is used by other OpenShift nodes to communicate with the node. Storage data traffic will use the interface attached to this IP. This must be the IP address and not the hostname

because, in an OpenShift environment, Heketi considers this to be the endpoint too.

- **devices:** Name of each disk to be added

Edit the topology file based on the Red Hat Gluster Storage pod hostname under the **node.hostnames.manage** section and **node.hostnames.storage** section with the IP address. For simplicity, the `/usr/share/heketi/topology-sample.json` file only sets up 4 nodes with 8 drives each.

4. Execute the following command on the client to deploy the heketi and Red Hat Gluster Storage pods:

```
# cns-deploy -n <namespace> --daemonset-label <NODE_LABEL> -g
topology.json
```

For example:

```
# cns-deploy -n storage-project-2 --daemonset-label glusterfs2 -g
topology.json
Welcome to the deployment tool for GlusterFS on Kubernetes and
OpenShift.
```

Before getting started, this script has some requirements of the execution environment and of the container platform that you should verify.

The client machine that will run this script must have:

- * Administrative access to an existing Kubernetes or OpenShift cluster
- * Access to a python interpreter 'python'
- * Access to the heketi client 'heketi-cli'

Each of the nodes that will host GlusterFS must also have appropriate firewall

rules for the required GlusterFS ports:

- * 2222 - sshd (if running GlusterFS in a pod)
- * 24007 - GlusterFS Daemon
- * 24008 - GlusterFS Management
- * 49152 to 49251 - Each brick for every volume on the host requires its own port. For every new brick, one new port will be used starting at 49152. We recommend a default range of 49152-49251 on each host, though you can adjust this to fit your needs.

In addition, for an OpenShift deployment you must:

- * Have 'cluster_admin' role on the administrative account doing the deployment
- * Add the 'default' and 'router' Service Accounts to the 'privileged' SCC
- * Have a router deployed that is configured to allow apps to access services running in the cluster

Do you wish to proceed with deployment?

[Y]es, [N]o? [Default: Y]: Y

Using OpenShift CLI.

NAME	STATUS	AGE
storage-project-2	Active	2m

Using namespace "storage-project-2".

Checking that heketi pod is not running ... OK

template "deploy-heketi" created

serviceaccount "heketi-service-account" created

template "heketi" created

template "glusterfs" created

role "edit" added: "system:serviceaccount:storage-project-2:heketi-service-account"

node "192.168.35.5" labeled

node "192.168.35.6" labeled

node "192.168.35.7" labeled

daemonset "glusterfs" created

Waiting for GlusterFS pods to start ... OK

service "deploy-heketi" created

route "deploy-heketi" created

deploymentconfig "deploy-heketi" created

Waiting for deploy-heketi pod to start ... OK

Creating cluster ... ID: fde139c21b0afcb6206bf272e0df1590

Creating node 192.168.35.5 ... ID: 0768a1ee35dce4cf707c7a1e9caa3d2a

Adding device /dev/vdc ... OK

Adding device /dev/vdd ... OK

Adding device /dev/vde ... OK

Adding device /dev/vdf ... OK

Creating node 192.168.35.6 ... ID: 63966f6fffd48c1980c4a2d03abeedd04

Adding device /dev/vdc ... OK

Adding device /dev/vdd ... OK

Adding device /dev/vde ... OK

Adding device /dev/vdf ... OK

Creating node 192.168.35.7 ... ID: de129c099193aaff2c64dca825f33558

Adding device /dev/vdc ... OK

Adding device /dev/vdd ... OK

Adding device /dev/vde ... OK

Adding device /dev/vdf ... OK

heketi topology loaded.

Saving heketi-storage.json

secret "heketi-storage-secret" created

endpoints "heketi-storage-endpoints" created

service "heketi-storage-endpoints" created

job "heketi-storage-copy-job" created

deploymentconfig "deploy-heketi" deleted

route "deploy-heketi" deleted

service "deploy-heketi" deleted

job "heketi-storage-copy-job" deleted

pod "deploy-heketi-1-d0qrs" deleted

secret "heketi-storage-secret" deleted

service "heketi" created

route "heketi" created

deploymentconfig "heketi" created

```
Waiting for heketi pod to start ... OK
heketi is now running.
Ready to create and provide GlusterFS volumes.
```



NOTE

For more information on the cns-deploy commands, see to the man page of the cns-deploy.

```
# cns-deploy --help
```

5. Verify that Red Hat Openshift Container Storage is deployed and working as expected in the new project with the new daemonSet label by executing the following command:

```
# oc get ds
```

For example:

```
# oc get ds
NAME          DESIRED   CURRENT   READY   NODE-SELECTOR
AGE
glusterfs     3         3         3       storagenode=glusterfs2
8m
```

1.2. REDUCING STORAGE CAPACITY

Heketi also supports the reduction of storage capacity. You can reduce storage by deleting devices, nodes, and clusters. These requests can only be performed by using the Heketi CLI or the API. For information on using command line API, see Heketi API <https://github.com/heketi/heketi/wiki/API>.



NOTE

- The IDs can be retrieved by executing the `heketi-cli topology info` command.

```
# heketi-cli topology info
```

- The **heketidbstorage** volume cannot be deleted as it contains the heketi database.

1.2.1. Deleting Volumes

You can delete the volume using the following Heketi CLI command:

```
# heketi-cli volume delete <volume_id>
```

For example:

```
heketi-cli volume delete 12b2590191f571be9e896c7a483953c3
Volume 12b2590191f571be9e896c7a483953c3 deleted
```

1.2.2. Deleting Device

Deleting the device deletes devices from heketi's topology. Devices that have bricks cannot be deleted. You must ensure they are free of bricks by disabling and removing devices.

1.2.2.1. Disabling and Enabling a Device

Disabling devices stops further allocation of bricks onto the device. You can disable devices using the following Heketi CLI command:

```
# heketi-cli device disable <device_id>
```

For example:

```
# heketi-cli device disable f53b13b9de1b5125691ee77db8bb47f4
Device f53b13b9de1b5125691ee77db8bb47f4 is now offline
```

If you want to re-enable the device, execute the following command. Enabling the device allows allocation of bricks onto the device.

```
# heketi-cli device enable <device_id>
```

For example:

```
# heketi-cli device enable f53b13b9de1b5125691ee77db8bb47f4
Device f53b13b9de1b5125691ee77db8bb47f4 is now online
```

1.2.2.2. Removing and Deleting the Device

Removing devices moves existing bricks from the device to other devices. This helps in ensuring the device is free of bricks. A device can be removed only after disabling it.

1. Remove device using the following command:

```
# heketi-cli device remove <device_id>
```

For example:

```
heketi-cli device remove e9ef1d9043ed3898227143add599e1f9
Device e9ef1d9043ed3898227143add599e1f9 is now removed
```

2. Delete the device using the following command:

```
# heketi-cli device delete <device_id>
```

For example:

```
heketi-cli device delete 56912a57287d07fad0651ba0003cf9aa
Device 56912a57287d07fad0651ba0003cf9aa deleted
```

The only way to reuse a deleted device is by adding the device to heketi's topology again.

1.2.2.3. Replacing a Device

Heketi does not allow one-to-one replacement of a device with another. However, in case of a failed device, follow the example below for the sequence of operations that are required to replace a failed device.

1. Locate the device that has failed using the following command:

```
# heketi-cli topology info

...
...
...
Nodes:
Node Id: 8faade64a9c8669de204b66bc083b10d
...
...
...
          Id:a811261864ee190941b17c72809a5001    Name:/dev/vdc
State:online    Size (GiB):499    Used (GiB):281    Free (GiB):218
          Bricks:

Id:34c14120bef5621f287951bcdfa774fc    Size (GiB):280    Path:
/var/lib/heketi/mounts/vg_a811261864ee190941b17c72809a5001/brick_34c
14120bef5621f287951bcdfa774fc/brick
...
...
...
```

The example below illustrates the sequence of operations that are required to replace a failed device. The example uses device ID **a811261864ee190941b17c72809a5001** which belongs to node with id **8faade64a9c8669de204b66bc083b10d** as.

2. Add a new device preferably to the same node as the device being replaced.

```
# heketi-cli device add --name /dev/vdd --node
8faade64a9c8669de204b66bc083b10d
Device added successfully
```

3. Disable the failed device.

```
# heketi-cli device disable a811261864ee190941b17c72809a5001
Device a811261864ee190941b17c72809a5001 is now offline
```

4. Remove the failed device.

```
# heketi-cli device remove a811261864ee190941b17c72809a5001
Device a811261864ee190941b17c72809a5001 is now removed
```

At this stage, the bricks are migrated from the failed device. Heketi chooses a suitable device based on the brick allocation algorithm. As a result, there is a possibility that all the bricks might not be migrated to the new added device.

5. Delete the failed device.


```
# heketi-cli device delete a811261864ee190941b17c72809a5001
Device a811261864ee190941b17c72809a5001 deleted
```

6. Before repeating the above sequence of steps on another device, you must wait for the self-heal operation to complete. You can verify that the self-heal operation completed when the Number of entries value returns a 0 value.

```
# oc rsh <any_gluster_pod_name>
for each in $(gluster volume list) ; do gluster vol heal $each info
| grep "Number of entries:" ; done
Number of entries: 0
Number of entries: 0
Number of entries: 0
```

1.2.3. Deleting Node

Nodes that have devices added to it cannot be deleted. To delete the node, the devices that are associated with the node have to be deleted. Disabling and removing the node ensures all the underlying devices are removed too. Once the node is removed, all the devices in it can be deleted and finally the node can be deleted

1.2.3.1. Disabling and Enabling a Node

Disabling node stops further allocation of bricks to all the devices associated to the node. You can disable nodes using the following Heketi CLI command:

```
# heketi-cli node disable <node_id>
```

For example:

```
heketi-cli node disable 5f0af88b968ed1f01bf959fe4fe804dc
Node 5f0af88b968ed1f01bf959fe4fe804dc is now offline
```

If you want to re-enable the node, execute the following command.

```
# heketi-cli node enable <node_id>
```

For example:

```
heketi-cli node enable 5f0af88b968ed1f01bf959fe4fe804dc
Node 5f0af88b968ed1f01bf959fe4fe804dc is now online
```

1.2.3.2. Removing and Deleting the Node

Removing nodes moves existing bricks from all the devices in the node to other devices in the cluster. This helps in ensuring all the device in the node is free of bricks. A device can be removed only after disabling it.

1. To remove the node execute the following command:

```
# heketi-cli node remove <node_id>
```

For example:

```
heketi-cli node remove 5f0af88b968ed1f01bf959fe4fe804dc
Node 5f0af88b968ed1f01bf959fe4fe804dc is now removed
```

2. Delete the devices associated with the node by executing the following command as the nodes that have devices associated with it cannot be deleted:

```
# heketi-cli device delete <device_id>
```

For example:

```
heketi-cli device delete 56912a57287d07fad0651ba0003cf9aa
Device 56912a57287d07fad0651ba0003cf9aa deleted
```

Execute the command for every device on the node.

3. Delete the node using the following command:

```
# heketi-cli node delete <node_id>
```

For example:

```
heketi-cli node delete 5f0af88b968ed1f01bf959fe4fe804dc
Node 5f0af88b968ed1f01bf959fe4fe804dc deleted
```

Deleting the node deletes the node from the heketi topology. The only way to reuse a deleted node is by adding the node to heketi's topology again

1.2.3.3. Replacing a Node

Heketi does not allow one-to-one replacement of a node with another. However, in case of a failed node, follow the example below for the sequence of operations that are required to replace a failed node and its respective devices.

1. Locate the node that has failed using the following command:

```
# heketi-cli topology info

...
...
...
Nodes:
Node Id: 8faade64a9c8669de204b66bc083b10d
...
...
...
State:online      Id:a811261864ee190941b17c72809a5001  Name:/dev/vdc
Size (GiB):499    Used (GiB):281      Free (GiB):218
Bricks:

Id:34c14120bef5621f287951bcdfa774fc  Size (GiB):280      Path:
/var/lib/heketi/mounts/vg_a811261864ee190941b17c72809a5001/brick_34c
14120bef5621f287951bcdfa774fc/brick
```

```
...
...
...
```

The example below illustrates the sequence of operations that are required to replace a failed node. The example uses node ID 8faade64a9c8669de204b66bc083b10d.

2. Add a new node, preferably that has the same devices as the node being replaced.

```
# heketi-cli node add --zone=1 --
cluster=597fceb5d6c876b899e48f599b988f54 --management-host-
name=node4.example.com --storage-host-name=192.168.10.104

# heketi-cli device add --name /dev/vdd --node
8faade64a9c8669de204b66bc083b10d

Node and device added successfully
```

3. Disable the failed node.

```
# heketi-cli node disable 8faade64a9c8669de204b66bc083b10d
Node 8faade64a9c8669de204b66bc083b10d is now offline
```

4. Remove the failed node.

```
# heketi-cli node remove 8faade64a9c8669de204b66bc083b10d
Node 8faade64a9c8669de204b66bc083b10d is now removed
```

At this stage, the bricks are migrated from the failed node. Heketi chooses a suitable device based on the brick allocation algorithm.

5. Delete the devices associated with the node by executing the following command as the nodes that have devices associated with it cannot be deleted:

```
# heketi-cli device delete <device_id>
```

For example:

```
heketi-cli device delete 56912a57287d07fad0651ba0003cf9aa
Device 56912a57287d07fad0651ba0003cf9aa deleted
```

Execute the command for every device on the node.

6. Delete the failed node.

```
# heketi-cli node delete 8faade64a9c8669de204b66bc083b10d
Node 8faade64a9c8669de204b66bc083b10d deleted
```

1.2.4. Deleting Clusters

You can delete the cluster using the following Heketi CLI command:

**NOTE**

Before a cluster is deleted, ensure that all the nodes inside the cluster are deleted.

```
# heketi-cli cluster delete <cluster_id>
```

For example:

```
heketi-cli cluster delete 0e949d91c608d13fd3fc4e96f798a5b1  
Cluster 0e949d91c608d13fd3fc4e96f798a5b1 deleted
```

CHAPTER 2. OPERATIONS ON A RED HAT GLUSTER STORAGE POD IN AN OPENSIFT ENVIRONMENT

This chapter lists out the various operations that can be performed on a Red Hat Gluster Storage pod (gluster pod):

1. To list the pods, execute the following command :

```
# oc get pods -n <storage_project_name>
```

For example:

```
# oc get pods -n storage-project
NAME                                     READY
STATUS   RESTARTS   AGE
storage-project-router-1-v89qc         1/1
Running   0          1d
glusterfs-dc-node1.example.com         1/1
Running   0          1d
glusterfs-dc-node2.example.com         1/1
Running   1          1d
glusterfs-dc-node3.example.com         1/1
Running   0          1d
heketi-1-k1u14                         1/1
Running   0          23m
```

Following are the gluster pods from the above example:

```
glusterfs-dc-node1.example.com
glusterfs-dc-node2.example.com
glusterfs-dc-node3.example.com
```



NOTE

The topology.json file will provide the details of the nodes in a given Trusted Storage Pool (TSP) . In the above example all the 3 Red Hat Gluster Storage nodes are from the same TSP.

2. To enter the gluster pod shell, execute the following command:

```
# oc rsh <gluster_pod_name> -n <storage_project_name>
```

For example:

```
# oc rsh glusterfs-dc-node1.example.com -n storage-project
sh-4.2#
```

3. To get the peer status, execute the following command:

```
# gluster peer status
```

For example:

```
# gluster peer status

Number of Peers: 2

Hostname: node2.example.com
Uuid: 9f3f84d2-ef8e-4d6e-aa2c-5e0370a99620
State: Peer in Cluster (Connected)
Other names:
node1.example.com

Hostname: node3.example.com
Uuid: 38621acd-eb76-4bd8-8162-9c2374affbbd
State: Peer in Cluster (Connected)
```

4. To list the gluster volumes on the Trusted Storage Pool, execute the following command:

```
# gluster volume info
```

For example:

```
Volume Name: heketidbstorage
Type: Distributed-Replicate
Volume ID: 2fa53b28-121d-4842-9d2f-dce1b0458fda
Status: Started
Number of Bricks: 2 x 3 = 6
Transport-type: tcp
Bricks:
Brick1:
192.168.121.172:/var/lib/heketi/mounts/vg_1be433737b71419dc9b395e221
255fb3/brick_c67fb97f74649d990c5743090e0c9176/brick
Brick2:
192.168.121.233:/var/lib/heketi/mounts/vg_0013ee200cdefaeb6dfedd28e5
0fd261/brick_6ebf1ee62a8e9e7a0f88e4551d4b2386/brick
Brick3:
192.168.121.168:/var/lib/heketi/mounts/vg_e4b32535c55c88f9190da7b7ef
d1fcab/brick_df5db97aa002d572a0fec6bcf2101aad/brick
Brick4:
192.168.121.233:/var/lib/heketi/mounts/vg_0013ee200cdefaeb6dfedd28e5
0fd261/brick_acc82e56236df912e9a1948f594415a7/brick
Brick5:
192.168.121.168:/var/lib/heketi/mounts/vg_e4b32535c55c88f9190da7b7ef
d1fcab/brick_65dceb1f749ec417533ddeae9535e8be/brick
Brick6:
192.168.121.172:/var/lib/heketi/mounts/vg_7ad961dbd24e16d62cabe10fd8
bf8909/brick_f258450fc6f025f99952a6edea203859/brick
Options Reconfigured:
performance.readdir-ahead: on

Volume Name: vol_9e86c0493f6b1be648c9deee1dc226a6
Type: Distributed-Replicate
Volume ID: 940177c3-d866-4e5e-9aa0-fc9be94fc0f4
Status: Started
Number of Bricks: 2 x 3 = 6
```

```

Transport-type: tcp
Bricks:
Brick1:
192.168.121.168:/var/lib/heketi/mounts/vg_3fa141bf2d09d30b899f2f260c
494376/brick_9fb4a5206bdd8ac70170d00f304f99a5/brick
Brick2:
192.168.121.172:/var/lib/heketi/mounts/vg_7ad961dbd24e16d62cabe10fd8
bf8909/brick_dae2422d518915241f74fd90b426a379/brick
Brick3:
192.168.121.233:/var/lib/heketi/mounts/vg_5c6428c439eb6686c5e4cee565
32bacf/brick_b3768ba8e80863724c9ec42446ea4812/brick
Brick4:
192.168.121.172:/var/lib/heketi/mounts/vg_7ad961dbd24e16d62cabe10fd8
bf8909/brick_0a13958525c6343c4a7951acec199da0/brick
Brick5:
192.168.121.168:/var/lib/heketi/mounts/vg_17fbc98d84df86756e7826326f
b33aa4/brick_af42af87ad87ab4f01e8ca153abbbbee9/brick
Brick6:
192.168.121.233:/var/lib/heketi/mounts/vg_5c6428c439eb6686c5e4cee565
32bacf/brick_ef41e04ca648efaf04178e64d25dbdcdb/brick
Options Reconfigured:
performance.readdir-ahead: on

```

5. To get the volume status, execute the following command:

```
# gluster volume status <volname>
```

For example:

```

# gluster volume status vol_9e86c0493f6b1be648c9deee1dc226a6

Status of volume: vol_9e86c0493f6b1be648c9deee1dc226a6
Gluster process                                TCP Port  RDMA Port
Online  Pid
-----
-----
Brick 192.168.121.168:/var/lib/heketi/mounts/v
g_3fa141bf2d09d30b899f2f260c494376/brick_9f
b4a5206bdd8ac70170d00f304f99a5/brick      49154      0          Y
3462
Brick 192.168.121.172:/var/lib/heketi/mounts/v
g_7ad961dbd24e16d62cabe10fd8bf8909/brick_da
e2422d518915241f74fd90b426a379/brick      49154      0          Y
115939
Brick 192.168.121.233:/var/lib/heketi/mounts/v
g_5c6428c439eb6686c5e4cee56532bacf/brick_b3
768ba8e80863724c9ec42446ea4812/brick      49154      0          Y
116134
Brick 192.168.121.172:/var/lib/heketi/mounts/v
g_7ad961dbd24e16d62cabe10fd8bf8909/brick_0a
13958525c6343c4a7951acec199da0/brick      49155      0          Y
115958
Brick 192.168.121.168:/var/lib/heketi/mounts/v
g_17fbc98d84df86756e7826326fb33aa4/brick_af
42af87ad87ab4f01e8ca153abbbbee9/brick      49155      0          Y

```

```

3481
Brick 192.168.121.233:/var/lib/heketi/mounts/v
g_5c6428c439eb6686c5e4cee56532bacf/brick_ef
41e04ca648efaf04178e64d25dbdcb/brick          49155      0          Y
116153
NFS Server on localhost                          2049      0          Y
116173
Self-heal Daemon on localhost                    N/A       N/A        Y
116181
NFS Server on node1.example.com
2049      0          Y          3501
Self-heal Daemon on node1.example.com
N/A       N/A      Y          3509
NFS Server on 192.168.121.172                    2049      0
Y          115978
Self-heal Daemon on 192.168.121.172              N/A       N/A
Y          115986

```

```
Task Status of Volume vol_9e86c0493f6b1be648c9deee1dc226a6
```

```
-----
-----
```

```
There are no active volume tasks
```

- To use the snapshot feature, load the snapshot module using the following command on one of the nodes:

```
# modprobe dm_snapshot
```

IMPORTANT

Restrictions for using Snapshot

- After a snapshot is created, it must be accessed through the user-serviceable snapshots feature only. This can be used to copy the old versions of files into the required location.

Reverting the volume to a snapshot state is not supported and should never be done as it might damage the consistency of the data.

- On a volume with snapshots, volume changing operations, such as volume expansion, must not be performed.

- To take the snapshot of the gluster volume, execute the following command:

```
# gluster snapshot create <snapname> <volname>
```

For example:

```

# gluster snapshot create snap1 vol_9e86c0493f6b1be648c9deee1dc226a6

snapshot create: success: Snap snap1_GMT-2016.07.29-13.05.46 created
successfully

```


8. To list the snapshots, execute the following command:

```
# gluster snapshot list
```

For example:

```
# gluster snapshot list

snap1_GMT-2016.07.29-13.05.46
snap2_GMT-2016.07.29-13.06.13
snap3_GMT-2016.07.29-13.06.18
snap4_GMT-2016.07.29-13.06.22
snap5_GMT-2016.07.29-13.06.26
```

9. To delete a snapshot, execute the following command:

```
# gluster snap delete <snapname>
```

For example:

```
# gluster snap delete snap1_GMT-2016.07.29-13.05.46

Deleting snap will erase all the information about the snap. Do you
still want to continue? (y/n) y
snapshot delete: snap1_GMT-2016.07.29-13.05.46: snap removed
successfully
```

For more information about managing snapshots, see

https://access.redhat.com/documentation/en-us/red_hat_gluster_storage/3.4/html-single/administration_guide/#chap-Managing_Snapshots.

10. You can set up Red Hat Openshift Container Storage volumes for geo-replication to a non-Red Hat Openshift Container Storage remote site. Geo-replication uses a master–slave model. Here, the Red Hat Openshift Container Storage volume acts as the master volume. To set up geo-replication, you must run the geo-replication commands on gluster pods. To enter the gluster pod shell, execute the following command:

```
# oc rsh <gluster_pod_name> -n <storage_project_name>
```

For more information about setting up geo-replication, see

https://access.redhat.com/documentation/en-us/red_hat_gluster_storage/3.4/html/administration_guide/chap-managing_geo-replication.

11. Brick multiplexing is a feature that allows including multiple bricks into one process. This reduces resource consumption, allowing you to run more bricks than earlier with the same memory consumption.

Brick multiplexing is enabled by default from Container-Native Storage 3.6. If you want to turn it off, execute the following command:

```
# gluster volume set all cluster.brick-multiplex off
```

12. The **auto_unmount** option in glusterfs libfuse, when enabled, ensures that the file system is unmounted at FUSE server termination by running a separate monitor process that performs the unmount.

The GlusterFS plugin in Openshift enables the **auto_unmount** option for gluster mounts.

PART II. OPERATIONS

CHAPTER 3. CREATING PERSISTENT VOLUMES

OpenShift Container Platform clusters can be provisioned with [persistent storage](#) using GlusterFS.

Persistent volumes (PVs) and persistent volume claims (PVCs) can share volumes across a single project. While the GlusterFS-specific information contained in a PV definition could also be defined directly in a pod definition, doing so does not create the volume as a distinct cluster resource, making the volume more susceptible to conflicts.

Binding PVs by Labels and Selectors

Labels are an OpenShift Container Platform feature that support user-defined tags (key-value pairs) as part of an object's specification. Their primary purpose is to enable the arbitrary grouping of objects by defining identical labels among them. These labels can then be targeted by selectors to match all objects with specified label values. It is this functionality we will take advantage of to enable our PVC to bind to our PV.

You can use labels to identify common attributes or characteristics shared among volumes. For example, you can define the gluster volume to have a custom attribute (key) named **storage-tier** with a value of **gold** assigned. A claim will be able to select a PV with **storage-tier=gold** to match this PV.

More details for provisioning volumes in file-based storage is provided in [Section 3.1, “File Storage”](#). Similarly, further details for provisioning volumes in block-based storage is provided in [Section 3.2, “Block Storage”](#).

3.1. FILE STORAGE

File storage, also called file-level or file-based storage, stores data in a hierarchical structure. The data is saved in files and folders, and presented to both the system storing it and the system retrieving it in the same format. You can provision volumes either statically or dynamically for file-based storage.

3.1.1. Static Provisioning of Volumes

To enable persistent volume support in OpenShift and Kubernetes, few endpoints and a service must be created:

The sample glusterfs endpoint file (sample-gluster-endpoints.yaml) and the sample glusterfs service file (sample-gluster-service.yaml) are available at `/usr/share/heketi/templates/` directory.

The sample endpoints and services file will not be available for ansible deployments since `/usr/share/heketi/templates/` directory will not be created for such deployments.



NOTE

Ensure to copy the sample glusterfs endpoint file / glusterfs service file to a location of your choice and then edit the copied file. For example:

```
# cp /usr/share/heketi/templates/sample-gluster-endpoints.yaml
  /<path>/gluster-endpoints.yaml
```

1. To specify the endpoints you want to create, update the copied **sample-gluster-endpoints.yaml** file with the endpoints to be created based on the environment. Each Red Hat Gluster Storage trusted storage pool requires its own endpoint with the IP of the nodes in the trusted storage pool.

```
# cat sample-gluster-endpoints.yaml
apiVersion: v1
kind: Endpoints
metadata:
  name: glusterfs-cluster
subsets:
- addresses:
  - ip: 192.168.10.100
  ports:
  - port: 1
- addresses:
  - ip: 192.168.10.101
  ports:
  - port: 1
- addresses:
  - ip: 192.168.10.102
  ports:
  - port: 1
```

name: is the name of the endpoint

ip: is the ip address of the Red Hat Gluster Storage nodes.

2. Execute the following command to create the endpoints:

```
# oc create -f <name_of_endpoint_file>
```

For example:

```
# oc create -f sample-gluster-endpoints.yaml
endpoints "glusterfs-cluster" created
```

3. To verify that the endpoints are created, execute the following command:

```
# oc get endpoints
```

For example:

```
# oc get endpoints
NAME                                ENDPOINTS
AGE
storage-project-router
192.168.121.233:80,192.168.121.233:443,192.168.121.233:1936    2d
glusterfs-cluster
192.168.121.168:1,192.168.121.172:1,192.168.121.233:1        3s
heketi                            10.1.1.3:8080
2m
heketi-storage-endpoints
192.168.121.168:1,192.168.121.172:1,192.168.121.233:1        3m
```

4. Execute the following command to create a gluster service:

```
# oc create -f <name_of_service_file>
```

For example:

```
# cat sample-gluster-service.yaml
apiVersion: v1
kind: Service
metadata:
  name: glusterfs-cluster
spec:
  ports:
    - port: 1

# oc create -f sample-gluster-service.yaml
service "glusterfs-cluster" created
```

- To verify that the service is created, execute the following command:

```
# oc get service
```

For example:

```
# oc get service
```

NAME	CLUSTER-IP	EXTERNAL-IP	PORT(S)
AGE			
storage-project-router	172.30.94.109	<none>	
80/TCP,443/TCP,1936/TCP	2d		
glusterfs-cluster	172.30.212.6	<none>	1/TCP
5s			
heketi	172.30.175.7	<none>	8080/TCP
2m			
heketi-storage-endpoints	172.30.18.24	<none>	1/TCP
3m			



NOTE

The endpoints and the services must be created for each project that requires a persistent storage.

- Create a 100G persistent volume with Replica 3 from GlusterFS and output a persistent volume specification describing this volume to the file pv001.json:

```
$ heketi-cli volume create --size=100 --persistent-volume-
file=pv001.json
```

```
cat pv001.json
{
  "kind": "PersistentVolume",
  "apiVersion": "v1",
  "metadata": {
    "name": "glusterfs-f8c612ee",
    "creationTimestamp": null
  },
  "spec": {
```

```

    "capacity": {
      "storage": "100Gi"
    },
    "glusterfs": {
      "endpoints": "TYPE ENDPOINT HERE",
      "path": "vol_f8c612eea57556197511f6b8c54b6070"
    },
    "accessModes": [
      "ReadWriteMany"
    ],
    "persistentVolumeReclaimPolicy": "Retain"
  },
  "status": {}

```



IMPORTANT

You must manually add the **Labels** information to the .json file.

Following is the example YAML file for reference:

```

apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-storage-project-glusterfs1
  labels:
    storage-tier: gold
spec:
  capacity:
    storage: 12Gi
  accessModes:
    - ReadWriteMany
  persistentVolumeReclaimPolicy: Retain
  glusterfs:
    endpoints: TYPE END POINTS NAME HERE,
    path: vol_e6b77204ff54c779c042f570a71b1407

```

name: The name of the volume.

storage: The amount of storage allocated to this volume

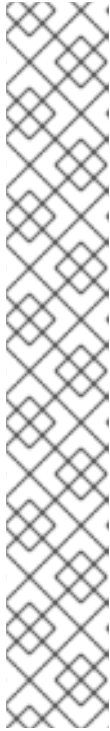
glusterfs: The volume type being used, in this case the glusterfs plug-in

endpoints: The endpoints name that defines the trusted storage pool created

path: The Red Hat Gluster Storage volume that will be accessed from the Trusted Storage Pool.

accessModes: accessModes are used as labels to match a PV and a PVC. They currently do not define any form of access control.

labels: Use labels to identify common attributes or characteristics shared among volumes. In this case, we have defined the gluster volume to have a custom attribute (key) named **storage-tier** with a value of **gold** assigned. A claim will be able to select a PV with **storage-tier=gold** to match this PV.



NOTE

- `heketi-cli` also accepts the endpoint name on the command line (`--persistent-volume-endpoint="TYPE ENDPOINT HERE"`). This can then be piped to `oc create -f -` to create the persistent volume immediately.
- If there are multiple Red Hat Gluster Storage trusted storage pools in your environment, you can check on which trusted storage pool the volume is created using the `heketi-cli volume list` command. This command lists the cluster name. You can then update the endpoint information in the `pv001.json` file accordingly.
- When creating a Heketi volume with only two nodes with the replica count set to the default value of three (replica 3), an error "No space" is displayed by Heketi as there is no space to create a replica set of three disks on three different nodes.
- If all the `heketi-cli` write operations (ex: volume create, cluster create..etc) fails and the read operations (ex: topology info, volume info ..etc) are successful, then the possibility is that the gluster volume is operating in read-only mode.

7. Edit the `pv001.json` file and enter the name of the endpoint in the endpoint's section:

```
cat pv001.json
{
  "kind": "PersistentVolume",
  "apiVersion": "v1",
  "metadata": {
    "name": "glusterfs-f8c612ee",
    "creationTimestamp": null,
    "labels": {
      "storage-tier": "gold"
    }
  },
  "spec": {
    "capacity": {
      "storage": "12Gi"
    },
    "glusterfs": {
      "endpoints": "glusterfs-cluster",
      "path": "vol_f8c612eea57556197511f6b8c54b6070"
    },
    "accessModes": [
      "ReadWriteMany"
    ],
    "persistentVolumeReclaimPolicy": "Retain"
  },
  "status": {}
}
```

8. Create a persistent volume by executing the following command:

```
# oc create -f pv001.json
```

For example:


```
# oc create -f pv001.json
persistentvolume "glusterfs-4fc22ff9" created
```

9. To verify that the persistent volume is created, execute the following command:

```
# oc get pv
```

For example:

```
# oc get pv
```

NAME	REASON	AGE	CAPACITY	ACCESSMODES	STATUS	CLAIM
glusterfs-4fc22ff9		4s	100Gi	RWX	Available	

10. Create a persistent volume claim file. For example:

```
# cat pvc.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: glusterfs-claim
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 100Gi
  selector:
    matchLabels:
      storage-tier: gold
```

11. Bind the persistent volume to the persistent volume claim by executing the following command:

```
# oc create -f pvc.yaml
```

For example:

```
# oc create -f pvc.yaml
persistentvolumeclaim "glusterfs-claim" created
```

12. To verify that the persistent volume and the persistent volume claim is bound, execute the following commands:

```
# oc get pv
# oc get pvc
```

For example:

```
# oc get pv
```

NAME	REASON	AGE	CAPACITY	ACCESSMODES	STATUS	CLAIM
glusterfs-4fc22ff9			100Gi	RWX	Bound	storage-
project/glusterfs-claim				1m		

```
# oc get pvc
```

NAME	ACCESSMODES	AGE	STATUS	VOLUME	CAPACITY	
glusterfs-claim			Bound	glusterfs-4fc22ff9	100Gi	RWX
11s						

13. The claim can now be used in the application:

For example:

```
# cat app.yaml

apiVersion: v1
kind: Pod
metadata:
  name: busybox
spec:
  containers:
    - image: busybox
      command:
        - sleep
        - "3600"
      name: busybox
      volumeMounts:
        - mountPath: /usr/share/busybox
          name: mypvc
  volumes:
    - name: mypvc
      persistentVolumeClaim:
        claimName: glusterfs-claim
```

```
# oc create -f app.yaml
pod "busybox" created
```

For more information about using the glusterfs claim in the application see, https://access.redhat.com/documentation/en-us/openshift_container_platform/3.10/html-single/configuring_clusters/#install-config-storage-examples-gluster-example.

14. To verify that the pod is created, execute the following command:

```
# oc get pods -n <storage_project_name>
```

For example:

```
# oc get pods -n storage-project
```

NAME	READY	STATUS	RESTARTS
AGE			

```

block-test-router-1-deploy      0/1      Running    0
4h
busybox                          1/1      Running    0
43s
glusterblock-provisioner-1-bjpz4 1/1      Running    0
4h
glusterfs-7l5xf                 1/1      Running    0
4h
glusterfs-hhxtk                 1/1      Running    3
4h
glusterfs-m4rbc                 1/1      Running    0
4h
heketi-1-3h9nb                  1/1      Running    0
4h

```

15. To verify that the persistent volume is mounted inside the container, execute the following command:

```
# oc rsh busybox
```

```

/ $ df -h
Filesystem                Size      Used Available Use% Mounted on
/dev/mapper/docker-253:0-1310998-
81732b5fd87c197f627a24bcd2777f12eec4ee937cc2660656908b2fa6359129
                                100.0G    34.1M    99.9G   0% /
tmpfs                      1.5G         0      1.5G   0% /dev
tmpfs                      1.5G         0      1.5G   0%
/sys/fs/cgroup
192.168.121.168:vol_4fc22ff934e531dec3830cfbcad1eeae
                                99.9G    66.1M    99.9G   0%
/usr/share/busybox
tmpfs                      1.5G         0      1.5G   0%
/run/secrets
/dev/mapper/vg_vagrant-lv_root
                                37.7G     3.8G    32.0G  11%
/dev/termination-log
tmpfs                      1.5G    12.0K     1.5G   0%
/var/run/secretgit s/kubernetes.io/serviceaccount

```

NOTE

If you encounter a permission denied error on the mount point, then refer to section Gluster Volume Security at: https://access.redhat.com/documentation/en-us/openshift_container_platform/3.10/html-single/configuring_clusters/#install-config-storage-examples-gluster-example.

3.1.2. Dynamic Provisioning of Volumes

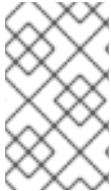
Dynamic provisioning enables you to provision a Red Hat Gluster Storage volume to a running application container without pre-creating the volume. The volume will be created dynamically as the claim request comes in, and a volume of exactly the same size will be provisioned to the application containers.

3.1.2.1. Configuring Dynamic Provisioning of Volumes

To configure dynamic provisioning of volumes, the administrator must define StorageClass objects that describe named "classes" of storage offered in a cluster. After creating a Storage Class, a secret for heketi authentication must be created before proceeding with the creation of persistent volume claim.

3.1.2.1.1. Creating Secret for Heketi Authentication

To create a secret for Heketi authentication, execute the following commands:



NOTE

If the **admin-key** value (secret to access heketi to get the volume details) was not set during the deployment of Red Hat Openshift Container Storage, then the following steps can be omitted.

1. Create an encoded value for the password by executing the following command:

```
# echo -n "<key>" | base64
```

where "key" is the value for "**admin-key**" that was created while deploying Red Hat Openshift Container Storage

For example:

```
# echo -n "mypassword" | base64
bXlwYXNzd29yZA==
```

2. Create a secret file. A sample secret file is provided below:

```
# cat glusterfs-secret.yaml

apiVersion: v1
kind: Secret
metadata:
  name: heketi-secret
  namespace: default
data:
  # base64 encoded password. E.g.: echo -n "mypassword" | base64
  key: bXlwYXNzd29yZA==
type: kubernetes.io/glusterfs
```

3. Register the secret on Openshift by executing the following command:

```
# oc create -f glusterfs-secret.yaml
secret "heketi-secret" created
```

3.1.2.1.2. Registering a Storage Class

When configuring a StorageClass object for persistent volume provisioning, the administrator must describe the type of provisioner to use and the parameters that will be used by the provisioner when it provisions a PersistentVolume belonging to the class.

1. To create a storage class execute the following command:

```
# cat > glusterfs-storageclass.yaml

apiVersion: storage.k8s.io/v1beta1
kind: StorageClass
metadata:
  name: gluster-container
provisioner: kubernetes.io/glusterfs
reclaimPolicy: Retain
parameters:
  resturl: "http://heketi-storage-project.cloudapps.mystorage.com"
  restuser: "admin"
  volumetype: "replicate:3"
  clusterid:
"630372ccdc720a92c681fb928f27b53f,796e6db1981f369ea0340913eeea4c9a"
  secretNamespace: "default"
  secretName: "heketi-secret"
  volumeoptions: "client.ssl on, server.ssl on"
  volumenameprefix: "test-vol"
allowVolumeExpansion: true
```

where,

resturl: Gluster REST service/Heketi service url which provision gluster volumes on demand. The general format must be IPaddress:Port and this is a mandatory parameter for GlusterFS dynamic provisioner. If Heketi service is exposed as a routable service in openshift/kubernetes setup, this can have a format similar to `http://heketi-storage-project.cloudapps.mystorage.com` where the fqdn is a resolvable heketi service url.

restuser : Gluster REST service/Heketi user who has access to create volumes in the trusted storage pool

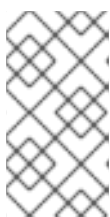
volumetype: It specifies the volume type that is being used.



NOTE

Distributed-Three-way replication is the only supported volume type.

clusterid: It is the ID of the cluster which will be used by Heketi when provisioning the volume. It can also be a list of comma-separated cluster IDs. This is an optional parameter.



NOTE

To get the cluster ID, execute the following command:

```
# heketi-cli cluster list
```

secretNamespace + secretName: Identification of Secret instance that contains the user password that is used when communicating with the Gluster REST service. These parameters are optional. Empty password will be used when both secretNamespace and secretName are

omitted.



NOTE

When the persistent volumes are dynamically provisioned, the Gluster plugin automatically creates an endpoint and a headless service in the name `gluster-dynamic-<claimname>`. This dynamic endpoint and service will be deleted automatically when the persistent volume claim is deleted.

volumeoptions: This is an optional parameter. It allows you to create glusterfs volumes with encryption enabled by setting the parameter to "client.ssl on, server.ssl on". For more information on enabling encryption, see [Chapter 8, Enabling Encryption](#).



NOTE

Do not add this parameter in the storageclass if encryption is not enabled.

volumenameprefix: This is an optional parameter. It depicts the name of the volume created by heketi. For more information see [Section 3.1.2.1.5, "\(Optional\) Providing a Custom Volume Name Prefix for Persistent Volumes"](#)



NOTE

The value for this parameter cannot contain ``_`` in the storageclass.

allowVolumeExpansion: To increase the PV claim value, ensure to set the **allowVolumeExpansion** parameter in the storageclass file to **true**. For more information, see [Section 3.1.2.1.7, "Expanding Persistent Volume Claim"](#).

2. To register the storage class to Openshift, execute the following command:

```
# oc create -f glusterfs-storageclass.yaml
storageclass "gluster-container" created
```

3. To get the details of the storage class, execute the following command:

```
# oc describe storageclass gluster-container

Name: gluster-container
IsDefaultClass: No
Annotations: <none>
Provisioner: kubernetes.io/glusterfs
Parameters: resturl=http://heketi-storage-
project.cloudapps.mystorage.com,restuser=admin,secretName=heketi-
secret,secretNamespace=default
No events.
```

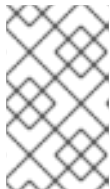
3.1.2.1.3. Creating a Persistent Volume Claim

To create a persistent volume claim execute the following commands:

1. Create a Persistent Volume Claim file. A sample persistent volume claim is provided below:

```
# cat glusterfs-pvc-claim1.yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: claim1
  annotations:
    volume.beta.kubernetes.io/storage-class: gluster-container
spec:
  persistentVolumeReclaimPolicy: Retain
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 5Gi
```

persistentVolumeReclaimPolicy: This is an optional parameter. When this parameter is set to "Retain" the underlying persistent volume is retained even after the corresponding persistent volume claim is deleted.



NOTE

When PVC is deleted, the underlying heketi and gluster volumes are not deleted if "persistentVolumeReclaimPolicy:" is set to "Retain". To delete the volume, you must use heketi cli and then delete the PV.

2. Register the claim by executing the following command:

```
# oc create -f glusterfs-pvc-claim1.yaml
persistentvolumeclaim "claim1" created
```

3. To get the details of the claim, execute the following command:

```
# oc describe pvc <claim_name>
```

For example:

```
# oc describe pvc claim1

Name: claim1
Namespace: default
StorageClass: gluster-container
Status: Bound
Volume: pvc-54b88668-9da6-11e6-965e-54ee7551fd0c
Labels: <none>
Capacity: 4Gi
Access Modes: RW0
No events.
```

3.1.2.1.4. Verifying Claim Creation

To verify if the claim is created, execute the following commands:

1. To get the details of the persistent volume claim and persistent volume, execute the following command:

```
# oc get pv,pvc
```

NAME	ACCESSMODES	RECLAIMPOLICY	STATUS	CLAIM	CAPACITY	AGE
pv/pvc-962aa6d1-bddb-11e6-be23-5254009fc65b					4Gi	
Delete		Bound		storage-project/claim1		3m

NAME	STATUS	VOLUME
pvc/claim1	Bound	pvc-962aa6d1-bddb-11e6-be23-5254009fc65b
4Gi	RWO	4m

2. To validate if the endpoint and the services are created as part of claim creation, execute the following command:

```
# oc get endpoints,service
```

NAME	ENDPOINTS
ep/storage-project-router	192.168.68.3:443,192.168.68.3:1936,192.168.68.3:80 28d
ep/gluster-dynamic-claim1	192.168.68.2:1,192.168.68.3:1,192.168.68.4:1 5m
ep/heketi	10.130.0.21:8080 21d
ep/heketi-storage-endpoints	192.168.68.2:1,192.168.68.3:1,192.168.68.4:1 25d

NAME	PORT(S)	AGE	CLUSTER-IP	EXTERNAL-IP
svc/storage-project-router	80/TCP,443/TCP,1936/TCP	28d	172.30.166.64	<none>
svc/gluster-dynamic-claim1		5m	172.30.52.17	<none> 1/TCP
svc/heketi	8080/TCP	21d	172.30.129.113	<none>
svc/heketi-storage-endpoints		25d	172.30.133.212	<none> 1/TCP

3.1.2.1.5. (Optional) Providing a Custom Volume Name Prefix for Persistent Volumes

You can provide a custom volume name prefix to the persistent volume that is created. By providing a custom volume name prefix, users can now easily search/filter the volumes based on:

- Any string that was provided as the field value of "volnameprefix" in the storageclass file.
- Persistent volume claim name.
- Project / Namespace name.

To set the name, ensure that you have added the parameter **volumenameprefix** to the storage class file. For more information, see [Section 3.1.2.1.2, “Registering a Storage Class”](#)



NOTE

The value for this parameter cannot contain `_` in the storageclass.

To verify if the custom volume name prefix is set, execute the following command:

```
# oc describe pv <pv_name>
```

For example:

```
# oc describe pv pvc-f92e3065-25e8-11e8-8f17-005056a55501
Name:          pvc-f92e3065-25e8-11e8-8f17-005056a55501
Labels:        <none>
Annotations:    Description=Gluster-Internal: Dynamically provisioned
PV
               gluster.kubernetes.io/heketi-volume-
id=027c76b24b1a3ce3f94d162f843529c8
               gluster.org/type=file
               kubernetes.io/createdby=heketi-dynamic-provisioner
               pv.beta.kubernetes.io/gid=2000
               pv.kubernetes.io/bound-by-controller=yes
               pv.kubernetes.io/provisioned-
by=kubernetes.io/glusterfs
               volume.beta.kubernetes.io/mount-options=auto_unmount
StorageClass:   gluster-container-prefix
Status:         Bound
Claim:          glusterfs/claim1
Reclaim Policy: Delete
Access Modes:   RW0
Capacity:       1Gi
Message:
Source:
  Type:          Glusterfs (a Glusterfs mount on the host that
shares a pod's lifetime)
  EndpointsName: glusterfs-dynamic-claim1
  Path:          test-vol_glusterfs_claim1_f9352e4c-25e8-11e8-
b460-005056a55501
  ReadOnly:      false
Events:         <none>
```

The value for **Path** will have the custom volume name prefix attached to the namespace and the claim name, which is "test-vol" in this case.

3.1.2.1.6. Using the Claim in a Pod

Execute the following steps to use the claim in a pod.

1. To use the claim in the application, for example

```
# cat app.yaml
```

```

apiVersion: v1
kind: Pod
metadata:
  name: busybox
spec:
  containers:
    - image: busybox
      command:
        - sleep
        - "3600"
      name: busybox
      volumeMounts:
        - mountPath: /usr/share/busybox
          name: mypvc
  volumes:
    - name: mypvc
      persistentVolumeClaim:
        claimName: claim1

```

```

# oc create -f app.yaml
pod "busybox" created

```

For more information about using the glusterfs claim in the application see, https://access.redhat.com/documentation/en-us/openshift_container_platform/3.10/html-single/configuring_clusters/#install-config-storage-examples-gluster-example.

2. To verify that the pod is created, execute the following command:

```

# oc get pods -n storage-project

```

NAME	READY	STATUS	
storage-project-router-1-at7tf	1/1	Running	0
13d			
busybox	1/1	Running	0
8s			
glusterfs-dc-192.168.68.2-1-hu28h	1/1	Running	0
7d			
glusterfs-dc-192.168.68.3-1-ytnlg	1/1	Running	0
7d			
glusterfs-dc-192.168.68.4-1-juqcq	1/1	Running	0
13d			
heketi-1-9r47c	1/1	Running	0
13d			

3. To verify that the persistent volume is mounted inside the container, execute the following command:

```

# oc rsh busybox

```

```

/ $ df -h

```

Filesystem	Size	Used	Available	Use%	Mounted on
/dev/mapper/docker-253:0-666733-					
38050a1d2cdb41dc00d60f25a7a295f6e89d4c529302fb2b93d8faa5a3205fb9					

	10.0G	33.8M	9.9G	0%	/
tmpfs	23.5G	0	23.5G	0%	/dev
tmpfs	23.5G	0	23.5G	0%	
/sys/fs/cgroup					
/dev/mapper/rhgs-root					
	17.5G	3.6G	13.8G	21%	
/run/secrets					
/dev/mapper/rhgs-root					
	17.5G	3.6G	13.8G	21%	
/dev/termination-log					
/dev/mapper/rhgs-root					
	17.5G	3.6G	13.8G	21%	
/etc/resolv.conf					
/dev/mapper/rhgs-root					
	17.5G	3.6G	13.8G	21%	
/etc/hostname					
/dev/mapper/rhgs-root					
	17.5G	3.6G	13.8G	21%	/etc/hosts
shm	64.0M	0	64.0M	0%	/dev/shm
192.168.68.2:vol_5b05cf2e5404afe614f8afa698792bae					
	4.0G	32.6M	4.0G	1%	
/usr/share/busybox					
tmpfs	23.5G	16.0K	23.5G	0%	
/var/run/secrets/kubernetes.io/serviceaccount					
tmpfs	23.5G	0	23.5G	0%	
/proc/kcore					
tmpfs	23.5G	0	23.5G	0%	
/proc/timer_stats					

3.1.2.1.7. Expanding Persistent Volume Claim

To increase the PV claim value, ensure to set the **allowVolumeExpansion** parameter in the storageclass file to **true**. For more information refer, [Section 3.1.2.1.2, “Registering a Storage Class”](#)



NOTE

You can also resize a PV via the OpenShift Container Platform 3.11 Web Console.

To expand the persistent volume claim value, execute the following commands:

1. If the feature gates **ExpandPersistentVolumes**, and the admissionconfig **PersistentVolumeClaimResize** are not enabled, then edit the master.conf file located at `/etc/origin/master/master-config.yaml` on the master to enable them. For example:

To enable feature gates **ExpandPersistentVolumes**

```
apiServerArguments:
  runtime-config:
    - apis/settings.k8s.io/v1alpha1=true
  storage-backend:
    - etcd3
  storage-media-type:
    - application/vnd.kubernetes.protobuf
  feature-gates:
```

```

- ExpandPersistentVolumes=true
controllerArguments:
  feature-gates:
- ExpandPersistentVolumes=true

```

To enable admissionconfig **PersistentVolumeClaimResize** add the following under admission config in the master-config file.

```

admissionConfig:
  pluginConfig:
    PersistentVolumeClaimResize:
      configuration:
        apiVersion: v1
        disable: false
        kind: DefaultAdmissionConfig

```

1. Restart the OpenShift master by running the following commands:

```

# /usr/local/bin/master-restart api
# /usr/local/bin/master-restart controllers

```

2. To check the existing persistent volume size, execute the following command on the app pod:

```
# oc rsh busybox
```

```
# df -h
```

For example:

```

# oc rsh busybox
/ # df -h

```

Filesystem	Size	Used	Available	Use%	Mounted on
/dev/mapper/docker-253:0-100702042-0fa327369e7708b67f0c632d83721cd9a5b39fd3a7b3218f3ff3c83ef4320ce7	10.0G	34.2M	9.9G	0%	/
tmpfs	15.6G	0	15.6G	0%	/dev
tmpfs	15.6G	0	15.6G	0%	
/sys/fs/cgroup					
/dev/mapper/rhel_dhcp47--150-root	50.0G	7.4G	42.6G	15%	
/dev/termination-log					
/dev/mapper/rhel_dhcp47--150-root	50.0G	7.4G	42.6G	15%	
/run/secrets					
/dev/mapper/rhel_dhcp47--150-root	50.0G	7.4G	42.6G	15%	
/etc/resolv.conf					
/dev/mapper/rhel_dhcp47--150-root	50.0G	7.4G	42.6G	15%	
/etc/hostname					
/dev/mapper/rhel_dhcp47--150-root	50.0G	7.4G	42.6G	15%	/etc/hosts
shm	64.0M	0	64.0M	0%	/dev/shm
10.70.46.177:test-vol_glusterfs_claim10_d3e15a8b-26b3-11e8-acdf-					

```

005056a55501
                2.0G      32.6M      2.0G      2%
/usr/share/busybox
tmpfs           15.6G      16.0K      15.6G      0%
/var/run/secrets/kubernetes.io/serviceaccount
tmpfs           15.6G           0      15.6G      0%
/proc/kcore
tmpfs           15.6G           0      15.6G      0%
/proc/timer_list
tmpfs           15.6G           0      15.6G      0%
/proc/timer_stats
tmpfs           15.6G           0      15.6G      0%
/proc/sched_debug
tmpfs           15.6G           0      15.6G      0% /proc/scsi
tmpfs           15.6G           0      15.6G      0%
/sys/firmware

```

In this example the persistent volume size is 2Gi

3. To edit the persistent volume claim value, execute the following command and edit the following storage parameter:

```

resources:
  requests:
    storage: <storage_value>

```

```
# oc edit pvc <claim_name>
```

For example, to expand the storage value to 20Gi:

```

# oc edit pvc claim3
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  annotations:
    pv.kubernetes.io/bind-completed: "yes"
    pv.kubernetes.io/bound-by-controller: "yes"
    volume.beta.kubernetes.io/storage-class: gluster-container2
    volume.beta.kubernetes.io/storage-provisioner:
kubernetes.io/glusterfs
  creationTimestamp: 2018-02-14T07:42:00Z
  name: claim3
  namespace: storage-project
  resourceVersion: "283924"
  selfLink: /api/v1/namespaces/storage-
project/persistentvolumeclaims/claim3
  uid: 8a9bb0df-115a-11e8-8cb3-005056a5a340
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 20Gi
  volumeName: pvc-8a9bb0df-115a-11e8-8cb3-005056a5a340
status:

```

```

accessModes:
- ReadWriteOnce
capacity:
  storage: 2Gi
phase: Bound

```

4. To verify, execute the following command on the app pod:

```
# oc rsh busybox
```

```
/ # df -h
```

For example:

```

# oc rsh busybox
# df -h

```

Filesystem	Size	Used	Available	Use%	Mounted on
/dev/mapper/docker-253:0-100702042-0fa327369e7708b67f0c632d83721cd9a5b39fd3a7b3218f3ff3c83ef4320ce7	10.0G	34.2M	9.9G	0%	/
tmpfs	15.6G	0	15.6G	0%	/dev
tmpfs	15.6G	0	15.6G	0%	
/sys/fs/cgroup					
/dev/mapper/rhel_dhcp47--150-root	50.0G	7.4G	42.6G	15%	
/dev/termination-log					
/dev/mapper/rhel_dhcp47--150-root	50.0G	7.4G	42.6G	15%	
/run/secrets					
/dev/mapper/rhel_dhcp47--150-root	50.0G	7.4G	42.6G	15%	
/etc/resolv.conf					
/dev/mapper/rhel_dhcp47--150-root	50.0G	7.4G	42.6G	15%	
/etc/hostname					
/dev/mapper/rhel_dhcp47--150-root	50.0G	7.4G	42.6G	15%	
shm	64.0M	0	64.0M	0%	/dev/shm
10.70.46.177:test-vol_glusterfs_claim10_d3e15a8b-26b3-11e8-acdf-005056a55501	20.0G	65.3M	19.9G	1%	
/usr/share/busybox					
tmpfs	15.6G	16.0K	15.6G	0%	
/var/run/secrets/kubernetes.io/serviceaccount					
tmpfs	15.6G	0	15.6G	0%	
/proc/kcore					
tmpfs	15.6G	0	15.6G	0%	
/proc/timer_list					
tmpfs	15.6G	0	15.6G	0%	
/proc/timer_stats					
tmpfs	15.6G	0	15.6G	0%	
/proc/sched_debug					
tmpfs	15.6G	0	15.6G	0%	/proc/scsi
tmpfs	15.6G	0	15.6G	0%	
/sys/firmware					

It is observed that the size is changed from 2Gi (earlier) to 20Gi.

3.1.2.1.8. Deleting a Persistent Volume Claim



NOTE

If the "persistentVolumeReclaimPolicy" parameter was set to "Retain" when registering the storageclass, the underlying PV and the corresponding volume remains even when a PVC is deleted.

1. To delete a claim, execute the following command:

```
# oc delete pvc <claim-name>
```

For example:

```
# oc delete pvc claim1
persistentvolumeclaim "claim1" deleted
```

2. To verify if the claim is deleted, execute the following command:

```
# oc get pvc <claim-name>
```

For example:

```
# oc get pvc claim1
No resources found.
```

When the user deletes a persistent volume claim that is bound to a persistent volume created by dynamic provisioning, apart from deleting the persistent volume claim, Kubernetes will also delete the persistent volume, endpoints, service, and the actual volume. Execute the following commands if this has to be verified:

- To verify if the persistent volume is deleted, execute the following command:

```
# oc get pv <pv-name>
```

For example:

```
# oc get pv pvc-962aa6d1-bddb-11e6-be23-5254009fc65b
No resources found.
```

- To verify if the endpoints are deleted, execute the following command:

```
# oc get endpoints <endpointname>
```

For example:

```
# oc get endpoints gluster-dynamic-claim1
No resources found.
```

- To verify if the service is deleted, execute the following command:

```
# oc get service <servicename>
```

For example:

```
# oc get service gluster-dynamic-claim1
No resources found.
```

3.1.3. Volume Security

Volumes come with a UID/GID of 0 (root). For an application pod to write to the volume, it should also have a UID/GID of 0 (root). With the volume security feature the administrator can now create a volume with a unique GID and the application pod can write to the volume using this unique GID

Volume security for statically provisioned volumes

To create a statically provisioned volume with a GID, execute the following command:

```
$ heketi-cli volume create --size=100 --persistent-volume-file=pv001.json
--gid=590
```

In the above command, a 100G persistent volume with a GID of 590 is created and the output of the persistent volume specification describing this volume is added to the pv001.json file.

For more information about accessing the volume using this GID, see

https://access.redhat.com/documentation/en-us/openshift_container_platform/3.10/html/configuring_clusters/persistent-storage-examples#install-config-storage-examples-gluster-example.

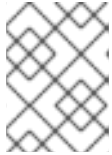
Volume security for dynamically provisioned volumes

Two new parameters, `gidMin` and `gidMax`, are introduced with dynamic provisioner. These values allow the administrator to configure the GID range for the volume in the storage class. To set up the GID values and provide volume security for dynamically provisioned volumes, execute the following commands:

1. Create a storage class file with the GID values. For example:

```
# cat glusterfs-storageclass.yaml

apiVersion: storage.k8s.io/v1beta1
kind: StorageClass
metadata:
  name:gluster-container
provisioner: kubernetes.io/glusterfs
parameters:
  resturl: "http://heketi-storage-project.cloudapps.mystorage.com"
  restuser: "admin"
  secretNamespace: "default"
  secretName: "heketi-secret"
  gidMin: "2000"
  gidMax: "4000"
```


**NOTE**

If the `gidMin` and `gidMax` value are not provided, then the dynamic provisioned volumes will have the GID between 2000 and 2147483647.

2. Create a persistent volume claim. For more information see, [Section 3.1.2.1.3, “Creating a Persistent Volume Claim”](#)
3. Use the claim in the pod. Ensure that this pod is non-privileged. For more information see, [Section 3.1.2.1.6, “Using the Claim in a Pod”](#)
4. To verify if the GID is within the range specified, execute the following command:

```
# oc rsh busybox
```

```
$ id
```

For example:

```
$ id
uid=1000060000 gid=0(root) groups=0(root),2001
```

where, 2001 in the above output is the allocated GID for the persistent volume, which is within the range specified in the storage class. You can write to this volume with the allocated GID.

**NOTE**

When the persistent volume claim is deleted, the GID of the persistent volume is released from the pool.

3.2. BLOCK STORAGE

Block storage allows the creation of high performance individual storage units. Unlike the traditional file storage capability that `glusterfs` supports, each storage volume/block device can be treated as an independent disk drive, so that each storage volume/block device can support an individual file system.

`gluster-block` is a distributed management framework for block devices. It aims to make Gluster-backed block storage creation and maintenance as simple as possible. `gluster-block` can provision block devices and export them as iSCSI LUN's across multiple nodes, and uses iSCSI protocol for data transfer as SCSI block/commands.

**NOTE**

Static provisioning of volumes is not supported for Block storage. Dynamic provisioning of volumes is the only method supported.

The recommended Red Hat Enterprise Linux (RHEL) version for block storage is RHEL-7.5.3.

Block volume expansion is not supported in Container-Native Storage 3.6.

3.2.1. Dynamic Provisioning of Volumes for Block Storage

Dynamic provisioning enables you to provision a Red Hat Gluster Storage volume to a running application container without pre-creating the volume. The volume will be created dynamically as the claim request comes in, and a volume of exactly the same size will be provisioned to the application containers.

3.2.1.1. Configuring Dynamic Provisioning of Volumes

To configure dynamic provisioning of volumes, the administrator must define StorageClass objects that describe named "classes" of storage offered in a cluster. After creating a Storage Class, a secret for heketi authentication must be created before proceeding with the creation of persistent volume claim.

3.2.1.1.1. Configuring Multipathing on all Initiators

To ensure the iSCSI initiator can communicate with the iSCSI targets and achieve HA using multipathing, execute the following steps on all the OpenShift nodes (iSCSI initiator) where the app pods are hosted:

1. To install initiator related packages on all the nodes where initiator has to be configured, execute the following command:

```
# yum install iscsi-initiator-utils device-mapper-multipath
```

2. To enable multipath, execute the following command:

```
# mpathconf --enable
```

3. Create and add the following content to the multipath.conf file:

```
# cat >> /etc/multipath.conf <<EOF
# LIO iSCSI
devices {
    device {
        vendor "LIO-ORG"
        user_friendly_names "yes" # names like mpatha
        path_grouping_policy "failover" # one path per
group
        hardware_handler "1 alua"
        path_selector "round-robin 0"
        failback immediate
        path_checker "tur"
        prio "alua"
        no_path_retry 120
    }
}
EOF
```

4. Execute the following commands to start multipath daemon and [re]load the multipath configuration:

```
# systemctl start multipathd
```

```
# systemctl reload multipathd
```

3.2.1.1.2. Creating Secret for Heketi Authentication

To create a secret for Heketi authentication, execute the following commands:



NOTE

If the **admin-key** value (secret to access heketi to get the volume details) was not set during the deployment of Red Hat OpenShift Container Storage, then the following steps can be omitted.

1. Create an encoded value for the password by executing the following command:

```
# echo -n "<key>" | base64
```

where “**key**” is the value for **admin-key** that was created while deploying CNS

For example:

```
# echo -n "mypassword" | base64
bXlwYXNzd29yZA==
```

2. Create a secret file. A sample secret file is provided below:

```
# cat glusterfs-secret.yaml

apiVersion: v1
kind: Secret
metadata:
  name: heketi-secret
  namespace: default
data:
  # base64 encoded password. E.g.: echo -n "mypassword" | base64
  key: bXlwYXNzd29yZA==
type: gluster.org/glusterblock
```

3. Register the secret on OpenShift by executing the following command:

```
# oc create -f glusterfs-secret.yaml
secret "heketi-secret" created
```

3.2.1.1.3. Registering a Storage Class

When configuring a StorageClass object for persistent volume provisioning, the administrator must describe the type of provisioner to use and the parameters that will be used by the provisioner when it provisions a PersistentVolume belonging to the class.

1. Create a storage class. A sample storage class file is presented below:

```
# cat > glusterfs-block-storageclass.yaml

apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
```

```

name: gluster-block
provisioner: gluster.org/glusterblock
reclaimPolicy: Retain
parameters:
  resturl: "http://heketi-storage-project.cloudapps.mystorage.com"
  restuser: "admin"
  restsecretnamespace: "default"
  restsecretname: "heketi-secret"
  hacount: "3"
  clusterids:
    "630372ccdc720a92c681fb928f27b53f,796e6db1981f369ea0340913eeea4c9a"
  chapauthenabled: "true"
  volumenameprefix: "test-vol"

```

where,

resturl: Gluster REST service/Heketi service url which provision gluster volumes on demand. The general format must be IPaddress:Port and this is a mandatory parameter for GlusterFS dynamic provisioner. If Heketi service is exposed as a routable service in openshift/kubernetes setup, this can have a format similar to `http://heketi-storage-project.cloudapps.mystorage.com` where the fqdn is a resolvable heketi service url.

restuser : Gluster REST service/Heketi user who has access to create volumes in the trusted storage pool

restsecretnamespace + restsecretname : Identification of Secret instance that contains user password to use when talking to Gluster REST service. These parameters are optional. Empty password will be used when both **restsecretnamespace** and **restsecretname** are omitted.

hacount: It is the count of the number of paths to the block target server. **hacount** provides high availability via multipathing capability of iSCSI. If there is a path failure, the I/Os will not be interrupted and will be served via another available paths.

clusterids: It is the ID of the cluster which will be used by Heketi when provisioning the volume. It can also be a list of comma-separated cluster IDs. This is an optional parameter.



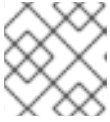
NOTE

To get the cluster ID, execute the following command:

```
# heketi-cli cluster list
```

chapauthenabled: If you want to provision block volume with CHAP authentication enabled, this value has to be set to true. This is an optional parameter.

volumenameprefix: This is an optional parameter. It depicts the name of the volume created by heketi. For more information see, [Section 3.2.1.1.6, “\(Optional\) Providing a Custom Volume Name Prefix for Persistent Volumes”](#)

**NOTE**

The value for this parameter cannot contain `_` in the storageclass.

2. To register the storage class to Openshift, execute the following command:

```
# oc create -f glusterfs-block-storageclass.yaml
storageclass "gluster-block" created
```

3. To get the details of the storage class, execute the following command:

```
# oc describe storageclass gluster-block
Name:          gluster-block
IsDefaultClass: No
Annotations:    <none>
Provisioner:    gluster.org/glusterblock
Parameters:
chapauthenabled=true,hacount=3,opmode=heketi,restsecretname=heketi-
secret,restsecretnamespace=default,resturl=http://heketi-storage-
project.cloudapps.mystorage.com,restuser=admin
Events:         <none>
```

3.2.1.1.4. Creating a Persistent Volume Claim

To create a persistent volume claim execute the following commands:

1. Create a Persistent Volume Claim file. A sample persistent volume claim is provided below:

```
# cat glusterfs-block-pvc-claim.yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: claim1
  annotations:
    volume.beta.kubernetes.io/storage-class: gluster-block
spec:
  persistentVolumeReclaimPolicy: Retain
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 5Gi
```

persistentVolumeReclaimPolicy: This is an optional parameter. When this parameter is set to "Retain" the underlying persistent volume is retained even after the corresponding persistent volume claim is deleted.

**NOTE**

When PVC is deleted, the underlying heketi and gluster volumes are not deleted if "persistentVolumeReclaimPolicy:" is set to "Retain". To delete the volume, you must use heketi cli and then delete the PV.

2. Register the claim by executing the following command:

```
# oc create -f glusterfs-block-pvc-claim.yaml
persistentvolumeclaim "claim1" created
```

3. To get the details of the claim, execute the following command:

```
# oc describe pvc <claim_name>
```

For example:

```
# oc describe pvc claim1

Name:          claim1
Namespace:     block-test
StorageClass:  gluster-block
Status:        Bound
Volume:        pvc-ee30ff43-7ddc-11e7-89da-5254002ec671
Labels:        <none>
Annotations:   control-plane.alpha.kubernetes.io/leader=
{"holderIdentity":"8d7fecb4-7dba-11e7-a347-0a580a830002",
"leaseDurationSeconds":15,"acquireTime":"2017-08-10T15:02:30Z",
"renewTime":"2017-08-10T15:02:58Z","lea...
  pv.kubernetes.io/bind-completed=yes
  pv.kubernetes.io/bound-by-controller=yes
  volume.beta.kubernetes.io/storage-class=gluster-block
  volume.beta.kubernetes.io/storage-
provisioner=gluster.org/glusterblock
Capacity:      5Gi
Access Modes:   RW0
Events:
  FirstSeen    LastSeen    Count   From          SubObjectPath  Type            Reason          Message
  ----
  1m           1m           1      gluster.org/glusterblock 8d7fecb4-7dba-11e7-a347-0a580a830002 Normal Provisioning External provisioner is provisioning volume for claim "block-test/claim1"
  1m           1m          18      persistentvolume-controller Normal ExternalProvisioning cannot find provisioner "gluster.org/glusterblock", expecting that a volume for the claim is provisioned either manually or via external software
  1m           1m           1      gluster.org/glusterblock 8d7fecb4-7dba-11e7-a347-0a580a830002 Normal ProvisioningSucceeded Successfully provisioned volume pvc-ee30ff43-7ddc-11e7-89da-5254002ec671
```

3.2.1.1.5. Verifying Claim Creation

To verify if the claim is created, execute the following commands:

1. To get the details of the persistent volume claim and persistent volume, execute the following command:

```
# oc get pv,pvc
```

NAME	ACCESSMODES	RECLAIMPOLICY	STATUS	CLAIM	CAPACITY
pv/pvc-ee30ff43-7ddc-11e7-89da-5254002ec671	Delete	Bound	block-test/claim1		5Gi
				gluster-block	RWO
					3m

NAME	CAPACITY	STATUS	VOLUME
pvc/claim1	5Gi	Bound	pvc-ee30ff43-7ddc-11e7-89da-5254002ec671
		RWO	gluster-block
			4m

3.2.1.1.6. (Optional) Providing a Custom Volume Name Prefix for Persistent Volumes

You can provide a custom volume name prefix to the persistent volume that is created. By providing a custom volume name prefix, users can now easily search/filter the volumes based on:

- Any string that was provided as the field value of "volnameprefix" in the storageclass file.
- Persistent volume claim name.
- Project / Namespace name.

To set the name, ensure that you have added the parameter **volumenameprefix** to the storage class file. For more information, refer [Section 3.2.1.1.3, "Registering a Storage Class"](#)



NOTE

The value for this parameter cannot contain `_` in the storageclass.

To verify if the custom volume name prefix is set, execute the following command:

```
# oc describe pv <pv_name>
```

For example:

```
# oc describe pv pvc-4e97bd84-25f4-11e8-8f17-005056a55501
Name:          pvc-4e97bd84-25f4-11e8-8f17-005056a55501
Labels:        <none>
Annotations:   AccessKey=glusterblk-67d422eb-7b78-4059-9c21-
a58e0eabe049-secret
                AccessKeyNs=glusterfs

Blockstring=url:http://172.31.251.137:8080,user:admin,secret:heketi-
secret,secretnamespace:glusterfs
                Description=Gluster-external: Dynamically
provisioned PV
                gluster.org/type=block
                gluster.org/volume-
id=cd37c089372040eba20904fb60b8c33e
                glusterBlkProvIdentity=gluster.org/glusterblock
                glusterBlockShare=test-
```

```

vol_glusterfs_bclaim1_4eab5a22-25f4-11e8-954d-0a580a830003
    kubernetes.io/createdby=heketi
    pv.kubernetes.io/provisioned-
by=gluster.org/glusterblock
    v2.0.0=v2.0.0
    StorageClass:    gluster-block-prefix
    Status:          Bound
    Claim:           glusterfs/bclaim1
    Reclaim Policy:  Delete
    Access Modes:    RW0
    Capacity:        5Gi
    Message:
    Source:
        Type:          ISCSI (an ISCSI Disk resource that is
attached to a kubelet's host machine and then exposed to the pod)
        TargetPortal:   10.70.46.177
        IQN:            iqn.2016-12.org.gluster-block:67d422eb-7b78-
4059-9c21-a58e0eabe049
        Lun:            0
        ISCSIInterface  default
        FSType:         xfs
        ReadOnly:       false
        Portals:        [10.70.46.142 10.70.46.4]
        DiscoveryCHAPAuth: false
        SessionCHAPAuth: true
        SecretRef:      {glusterblk-67d422eb-7b78-4059-9c21-
a58e0eabe049-secret }
        InitiatorName:  <none>
    Events:           <none>

```

The value for **glusterBlockShare** will have the custom volume name prefix attached to the namespace and the claim name, which is "test-vol" in this case.

3.2.1.1.7. Using the Claim in a Pod

Execute the following steps to use the claim in a pod.

1. To use the claim in the application, for example

```

# cat app.yaml

apiVersion: v1
kind: Pod
metadata:
  name: busybox
spec:
  containers:
    - image: busybox
      command:
        - sleep
        - "3600"
      name: busybox
      volumeMounts:
        - mountPath: /usr/share/busybox
          name: mypvc
  volumes:

```



```
- name: mypvc
  persistentVolumeClaim:
    claimName: claim1
```

```
# oc create -f app.yaml
pod "busybox" created
```

For more information about using the glusterfs claim in the application see, https://access.redhat.com/documentation/en-us/openshift_container_platform/3.10/html-single/configuring_clusters/#install-config-storage-examples-gluster-example.

2. To verify that the pod is created, execute the following command:

```
# oc get pods -n storage-project
```

NAME	READY	STATUS	RESTARTS
AGE			
block-test-router-1-deploy	0/1	Running	0
4h			
busybox	1/1	Running	0
43s			
glusterblock-provisioner-1-bj pz4	1/1	Running	0
4h			
glusterfs-7l5xf	1/1	Running	0
4h			
glusterfs-hhxtk	1/1	Running	3
4h			
glusterfs-m4rbc	1/1	Running	0
4h			
heketi-1-3h9nb	1/1	Running	0
4h			

3. To verify that the persistent volume is mounted inside the container, execute the following command:

```
# oc rsh busybox
```

```
/ # df -h
```

Filesystem	Size	Used	Available	Use%	Mounted on
/dev/mapper/docker-253:1-11438-39febd9d64f3a3594fc11da83d6cbaf5caf32e758eb9e2d7bdd798752130de7e	10.0G	33.9M	9.9G	0%	/
tmpfs	3.8G	0	3.8G	0%	/dev
tmpfs	3.8G	0	3.8G	0%	
/sys/fs/cgroup					
/dev/mapper/VolGroup00-LogVol00	7.7G	2.8G	4.5G	39%	
/dev/termination-log					
/dev/mapper/VolGroup00-LogVol00	7.7G	2.8G	4.5G	39%	
/run/secrets					
/dev/mapper/VolGroup00-LogVol00	7.7G	2.8G	4.5G	39%	
/etc/resolv.conf					

```

/dev/mapper/VolGroup00-LogVol00
              7.7G      2.8G      4.5G   39%
/etc/hostname
/dev/mapper/VolGroup00-LogVol00
              7.7G      2.8G      4.5G   39% /etc/hosts
shm           64.0M      0        64.0M   0% /dev/shm
/dev/mpatha   5.0G      32.2M      5.0G    1%
/usr/share/busybox
tmpfs         3.8G      16.0K      3.8G    0%
/var/run/secrets/kubernetes.io/serviceaccount
tmpfs         3.8G      0        3.8G    0%
/proc/kcore
tmpfs         3.8G      0        3.8G    0%
/proc/timer_list
tmpfs         3.8G      0        3.8G    0%
/proc/timer_stats
tmpfs         3.8G      0        3.8G    0%
/proc/sched_debug

```

3.2.1.1.8. Deleting a Persistent Volume Claim



NOTE

If the "persistentVolumeReclaimPolicy" parameter was set to "Retain" when registering the storageclass, the underlying PV and the corresponding volume remains even when a PVC is deleted.

1. To delete a claim, execute the following command:

```
# oc delete pvc <claim-name>
```

For example:

```
# oc delete pvc claim1
persistentvolumeclaim "claim1" deleted
```

2. To verify if the claim is deleted, execute the following command:

```
# oc get pvc <claim-name>
```

For example:

```
# oc get pvc claim1
No resources found.
```

When the user deletes a persistent volume claim that is bound to a persistent volume created by dynamic provisioning, apart from deleting the persistent volume claim, Kubernetes will also delete the persistent volume, endpoints, service, and the actual volume. Execute the following commands if this has to be verified:

- To verify if the persistent volume is deleted, execute the following command:

```
# oc get pv <pv-name>
```

■

For example:

```
# oc get pv pvc-962aa6d1-bddb-11e6-be23-5254009fc65b
No resources found.
```

Next step: If you are installing Red Hat Openshift Container Storage 3.10, and you want to use block storage as the backend storage for logging and metrics, proceed to [Chapter 7, *Gluster Block Storage as Backend for Logging and Metrics*](#).

3.2.2. Replacing a Block on Block Storage

If you want to replace a block from a node that is out of resource or is faulty, it can be replaced to a new node.

Execute the following commands

1. Execute the following command to fetch the zone and cluster info from heketi

```
# heketi-cli topology info --user=<user> --secret=<user key>
```

--user - heketi user

--secret - Secret key for a specified user

2. After obtaining the cluster id and zone id add a new node to heketi by executing the following command:



NOTE

Before adding the node, ensure the node is labeled as a glusterfs storage host by adding the label "glusterfs=storage-host", using the following command;

```
# oc label node <NODENAME> glusterfs=storage-host
```

```
# heketi-cli node add --zone=<zoneid> --cluster=<clusterid> --
management-host-name=<new hostname> --storage-host-name=<new node
ip> --user=<user> --secret=<user key>
```

--cluster - The cluster in which the node should reside

--management-host-name - Management hostname. This is the new node that has to be added.

--storage-host-name - Storage hostname.

--zone - The zone in which the node should reside

--user - heketi user.

--secret - Secret key for a specified user

For example:

```
heketi-cli node add --zone=1 --  
cluster=607204cb27346a221f39887a97cf3f90 --management-host-  
name=dhcp43-241.lab.eng.blr.redhat.com --storage-host-  
name=10.70.43.241 --user=admin --secret=adminkey
```

Node information:

```
Id: 2639c473a2805f6e19d45997bb18cb9c  
State: online  
Cluster Id: 607204cb27346a221f39887a97cf3f90  
Zone: 1  
Management Hostname dhcp43-241.lab.eng.blr.redhat.com  
Storage Hostname 10.70.43.241
```

3. Execute the following command to add the device

```
# heketi-cli device add --name=<device name> --node=<node id> --  
user=<user> --secret=<user key>
```

--name - Name of device to add

--node - Newly added node id

For example:

```
# heketi-cli device add --name=/dev/vdc --  
node=2639c473a2805f6e19d45997bb18cb9c --user=admin --secret=adminkey  
Device added successfully
```

4. After the new node and its associated devices are added to heketi, the faulty or unwanted node can be removed from heketi

To remove any node from heketi, follow this workflow:

- node disable (Disallow usage of a node by placing it offline)
- node replace (Removes a node and all its associated devices from Heketi)
- device delete (Deletes a device from Heketi node)
- node delete (Deletes a node from Heketi management)

5. Execute the following command to fetch the node list from heketi

```
#heketi-cli node list --user=<user> --secret=<user key>
```

For example:

```
# heketi-cli node list --user=admin --secret=adminkey  
Id:05746c562d6738cb5d7de149be1dac04  
Cluster:607204cb27346a221f39887a97cf3f90  
Id:ab37fc5aabb714eb8b09c9a868163df  
Cluster:607204cb27346a221f39887a97cf3f90  
Id:c513da1f9bda528a9fd6da7cb546a1ee  
Cluster:607204cb27346a221f39887a97cf3f90  
Id:e6ab1fe377a420b8b67321d9e60c1ad1
```

```
Cluster:607204cb27346a221f39887a97cf3f90
```

- Execute the following command to fetch the node info of the node, that has to be deleted from heketi:

```
# heketi-cli node info <nodeid> --user=<user> --secret=<user key>
```

For example:

```
# heketi-cli node info c513da1f9bda528a9fd6da7cb546a1ee --user=admin
--secret=adminkey
Node Id: c513da1f9bda528a9fd6da7cb546a1ee
State: online
Cluster Id: 607204cb27346a221f39887a97cf3f90
Zone: 1
Management Hostname: dhcp43-171.lab.eng.blr.redhat.com
Storage Hostname: 10.70.43.171
Devices:
Id:3a1e0717e6352a8830ab43978347a103    Name:/dev/vdc
State:online    Size (GiB):499    Used (GiB):100    Free (GiB):399
Bricks:1
Id:89a57ace1c3184826e1317fef785e6b7    Name:/dev/vdd
State:online    Size (GiB):499    Used (GiB):10    Free (GiB):489
Bricks:5
```

- Execute the following command to disable the node from heketi. This makes the node go offline:

```
# heketi-cli node disable <node-id> --user=<user> --secret=<user
key>
```

For example:

```
# heketi-cli node disable ab37fc5aabbd714eb8b09c9a868163df --
user=admin --secret=adminkey
Node ab37fc5aabbd714eb8b09c9a868163df is now offline
```

- Execute the following command to remove a node and all its associated devices from Heketi:

```
#heketi-cli node remove <node-id> --user=<user> --secret=<user key>
```

For example:

```
# heketi-cli node remove ab37fc5aabbd714eb8b09c9a868163df --
user=admin --secret=adminkey
Node ab37fc5aabbd714eb8b09c9a868163df is now removed
```

- Execute the following command to delete the devices from heketi node:

```
# heketi-cli device delete <device-id> --user=<user> --secret=<user
key>
```

For example:

```
# heketi-cli device delete 0fca78c3a94faabfbe5a5a9eef01b99c --
user=admin --secret=adminkey
Device 0fca78c3a94faabfbe5a5a9eef01b99c deleted
```

10. Execute the following command to delete a node from Heketi management:

```
#heketi-cli node delete <nodeid> --user=<user> --secret=<user key>
```

For example:

```
# heketi-cli node delete ab37fc5aabbd714eb8b09c9a868163df --
user=admin --secret=adminkey
Node ab37fc5aabbd714eb8b09c9a868163df deleted
```

11. Execute the following commands on any one of the gluster pods to replace the faulty node with the new node:

1. Execute the following command to get list of blockvolumes hosted under block-hosting-volume

```
# gluster-block list <block-hosting-volume> --json-pretty
```

2. Execute the following command to find out which all blockvolumes are hosted on the old node, with the help of info command

```
# gluster-block info <block-hosting-volume>/<block-volume> --
json-pretty
```

3. Execute the following command to replace the faulty node with the new node:

```
# gluster-block replace <volname/blockname> <old-node> <new-node>
[force]
```

For example:

```
{
  "NAME": "block",
  "CREATE SUCCESS": "192.168.124.73",
  "DELETE SUCCESS": "192.168.124.63",
  "REPLACE PORTAL SUCCESS ON": [
    "192.168.124.79"
  ],
  "RESULT": "SUCCESS"
}
```

Note: If the old node is down and does not come up again then you can force replace:

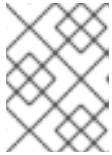
```
gluster-block replace sample/block 192.168.124.63 192.168.124.73
force --json-pretty
```

```
{
```

```

    "NAME": "block",
    "CREATE SUCCESS": "192.168.124.73",
    "DELETE FAILED (ignored)": "192.168.124.63",
    "REPLACE PORTAL SUCCESS ON": [
        "192.168.124.79"
    ],
    "RESULT": "SUCCESS"
}

```



NOTE

The next steps henceforth are to be executed only if the block that is to be replaced is still in use.

12. Logout of the old portal by executing the following command on the initiator:

```
# iscsiadm -m node -T <targetname> -p <old node> -u
```

For example:

```

# iscsiadm -m node -T iqn.2016-12.org.gluster-block:d6d18f43-8a74-
4b2c-a5b7-df1fa3f5bc9a -p 192.168.124.63 -u
Logging out of session [sid: 8, target: iqn.2016-12.org.gluster-
block:d6d18f43-8a74-4b2c-a5b7-df1fa3f5bc9a, portal:
192.168.124.63,3260]
Logout of [sid: 8, target: iqn.2016-12.org.gluster-block:d6d18f43-
8a74-4b2c-a5b7-df1fa3f5bc9a, portal: 192.168.124.63,3260]
successful.

```

13. To re-discover the new node execute the following command:

```
# iscsiadm -m discovery -t st -p <new node>
```

For example:

```

# iscsiadm -m discovery -t st -p 192.168.124.73
192.168.124.79:3260,1 iqn.2016-12.org.gluster-block:d6d18f43-8a74-
4b2c-a5b7-df1fa3f5bc9a
192.168.124.73:3260,2 iqn.2016-12.org.gluster-block:d6d18f43-8a74-
4b2c-a5b7-df1fa3f5bc9a

```

14. Login to the new portal by executing the following command:

```
# iscsiadm -m node -T <targetname> -p <new node ip> -l
```

For example:

```

# iscsiadm -m node -T iqn.2016-12.org.gluster-block:d6d18f43-8a74-
4b2c-a5b7-df1fa3f5bc9a -p 192.168.124.73 -l

```

15. To verify if the enabled hosting volume is replaced and running successfully, execute the following command on the initiator:

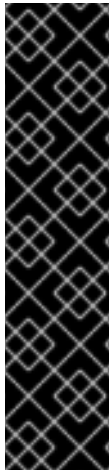
```
# ll /dev/disk/by-path/ip-* | grep <targetname> | grep <"new node  
ip">
```


CHAPTER 4. SHUTTING DOWN GLUSTER-BLOCK CLIENT NODES

Follow this procedure to shutdown gluster-block client nodes:

1. Evacuate the pods. For more information, refer https://access.redhat.com/documentation/en-us/openshift_container_platform/3.10/html-single/cluster_administration/#evacuating-pods-on-nodes
2. Ensure that no gluster block mounts exist in the system.
3. Reboot the nodes. For more information, refer https://access.redhat.com/documentation/en-us/openshift_container_platform/3.10/html-single/cluster_administration/#rebooting-nodes

CHAPTER 5. S3 COMPATIBLE OBJECT STORE IN A RED HAT OPENSIFT CONTAINER STORAGE ENVIRONMENT



IMPORTANT

Support for S3 compatible Object Store in Container-Native Storage is under technology preview. Technology Preview features are not fully supported under Red Hat service-level agreements (SLAs), may not be functionally complete, and are not intended for production use.

Tech Preview features provide early access to upcoming product innovations, enabling customers to test functionality and provide feedback during the development process.

As Red Hat considers making future iterations of Technology Preview features generally available, we will provide commercially reasonable efforts to resolve any reported issues that customers experience when using these features.

Object Store provides a system for data storage that enables users to access the same data, both as an object and as a file, thus simplifying management and controlling storage costs. The S3 API is the de facto standard for HTTP based access to object storage services.

5.1. SETTING UP S3 COMPATIBLE OBJECT STORE FOR RED HAT OPENSIFT CONTAINER STORAGE



NOTE

Ensure that `cns-deploy` package has been installed before setting up S3 Compatible Object Store. For more information on how to install `cns-deploy` package, see https://access.redhat.com/documentation/en-us/red_hat_openshift_container_storage/3.10/html-single/deployment_guide/#part-Appendix

Execute the following steps from the `/usr/share/heketi/templates/` directory to set up S3 compatible object store for Red Hat Openshift Container Storage:

1. (Optional): If you want to create a secret for heketi, then execute the following command:

```
# oc create secret generic heketi-${NAMESPACE}-admin-secret
--from-literal=key=${ADMIN_KEY} --type=kubernetes.io/glusterfs
```

For example:

```
# oc create secret generic heketi-storage-project-admin-secret
--from-literal=key= --type=kubernetes.io/glusterfs
```

1. Execute the following command to label the secret:

```
# oc label --overwrite secret heketi-${NAMESPACE}-admin-secret
glusterfs=s3-heketi-${NAMESPACE}-admin-secret
gluster-s3=heketi-${NAMESPACE}-admin-secret
```

For example:

```
# oc label --overwrite secret heketi-storage-project-admin-secret
glusterfs=s3-heketi-storage-project-admin-secret
gluster-s3=heketi-storage-project-admin-secret
```

2. Create a GlusterFS StorageClass file. Use the **HEKETI_URL** and **NAMESPACE** from the current setup and set a **STORAGE_CLASS** name.

```
# sed -e 's/${HEKETI_URL}/heketi-storage-
project.cloudapps.mystorage.com/g' -e 's/${STORAGE_CLASS}/gluster-
s3-store/g' -e 's/${NAMESPACE}/storage-project/g'
/usr/share/heketi/templates/gluster-s3-storageclass.yaml | oc create
-f -
```

For example:

```
# sed -e 's/${HEKETI_URL}/heketi-storage-
project.cloudapps.mystorage.com/g' -e 's/${STORAGE_CLASS}/gluster-
s3-store/g' -e 's/${NAMESPACE}/storage-project/g'
/usr/share/heketi/templates/gluster-s3-storageclass.yaml | oc create
-f -storageclass "gluster-s3-store" created
```

NOTE

- o You can run the following command to obtain the HEKETI_URL:

```
# oc get routes --all-namespaces | grep heketi
```

A sample output of the command is as follows:

```
glusterfs    heketi-storage
heketi-storage-
glusterfs.router.default.svc.cluster.local
heketi-storage    <all>          None
```

If there are multiple lines in the output then you can choose the most relevant one.

- o You can run the following command to obtain the NAMESPACE:

```
oc get project
```

A sample output of the command is as follows:

```
# oc project
Using project "glusterfs" on server
"master.example.com:8443"
```

where, glusterfs is the NAMESPACE.

3. Create the Persistent Volume Claims using the storage class.

```
# sed -e 's/${VOLUME_CAPACITY}/2Gi/g' -e
's/${STORAGE_CLASS}/gluster-s3-store/g'
/usr/share/heketi/templates/gluster-s3-pvcs.yaml | oc create -f -
```

For Example:

```
# sed -e 's/${VOLUME_CAPACITY}/2Gi/g' -e
's/${STORAGE_CLASS}/gluster-s3-store/g'
/usr/share/heketi/templates/gluster-s3-pvcs.yaml | oc create -f -
persistentvolumeclaim "gluster-s3-claim" created
persistentvolumeclaim "gluster-s3-meta-claim" created
```

Use the **STORAGE_CLASS** created from the previous step. Modify the **VOLUME_CAPACITY** as per the environment requirements. Wait till the PVC is bound. Verify the same using the following command:

```
# oc get pvc
NAME                                STATUS    VOLUME
CAPACITY  ACCESSMODES  AGE
gluster-s3-claim          Bound      pvc-0b7f75ef-9920-11e7-9309-
00151e000016    2Gi      RWX      2m
gluster-s3-meta-claim     Bound      pvc-0b87a698-9920-11e7-9309-
00151e000016    1Gi      RWX      2m
```

4. Start the glusters3 object storage service using the template:



NOTE

Set the **S3_ACCOUNT** name, **S3_USER** name, and **S3_PASSWORD**. **PVC** and **META_PVC** are obtained from the previous step.

```
# oc new-app /usr/share/heketi/templates/gluster-s3-template.yaml \
--param=S3_ACCOUNT=testvolume --param=S3_USER=adminuser \
--param=S3_PASSWORD=itsmine --param=PVC=gluster-s3-claim \
--param=META_PVC=gluster-s3-meta-claim
--> Deploying template "storage-project/gluster-s3" for
"/usr/share/heketi/templates/gluster-s3-template.yaml" to project
storage-project

gluster-s3
-----
Gluster s3 service template

* With parameters:
  * S3 Account Name=testvolume
  * S3 User=adminuser
  * S3 User Password=itsmine
  * Primary GlusterFS-backed PVC=gluster-s3-claim
  * Metadata GlusterFS-backed PVC=gluster-s3-meta-claim

--> Creating resources ...
service "gluster-s3-service" created
```

```

route "gluster-s3-route" created
deploymentconfig "gluster-s3-dc" created
--> Success
Run 'oc status' to view your app.
    
```

5. Execute the following command to verify if the S3 pod is up:

```

# oc get route
NAME                                HOST/PORT
PATH      SERVICES                    PORT      TERMINATION  WILDCARD
gluster-S3-route  gluster-s3-route-storage-
project.cloudapps.mystorage.com ... 1 more      gluster-s3-
service  <all>                          None
heketi      heketi-storage-project.cloudapps.mystorage.com ...
1 more      heketi                          <all>
    
```

5.2. OBJECT OPERATIONS

This section lists some of the object operation that can be performed:

- Get the URL of the route which provides S3 OS

```

# s3_storage_url=$(oc get routes | grep "gluster.*s3" | awk
'{print $2}')
    
```

NOTE

Ensure to download the s3curl tool from <https://aws.amazon.com/code/128>. This tool will be used for verifying the object operations.

- s3curl.pl requires Digest::HMAC_SHA1 and Digest::MD5. Install the perl-Digest-HMAC package to get this. You can install the perl-Digest-HMAC package by running this command:

```
# yum install perl-Digest-HMAC
```

- Update the s3curl.pl perl script with glusters3object url which was retrieved:

For example:

```
my @endpoints = ( 'glusters3object-storage-
project.cloudapps.mystorage.com' );
```

- To perform **PUT** operation of the bucket:

```
s3curl.pl --debug --id "testvolume:adminuser" --key "itsmine" --put
/dev/null -- -k -v http://$s3_storage_url/bucket1
```

- To perform **PUT** operation of the object inside the bucket:

```
s3curl.pl --debug --id "testvolume:adminuser" --key "itsmine" --put  
my_object.jpg -- -k -v -s  
http://$s3_storage_url/bucket1/my_object.jpg
```

- To verify listing of objects in the bucket:

```
s3curl.pl --debug --id "testvolume:adminuser" --key "itsmine" -- -k  
-v -s http://$s3_storage_url/bucket1/
```

CHAPTER 6. CLUSTER ADMINISTRATOR SETUP

Authentication

Set up the authentication using **AllowAll Authentication** method.

AllowAll Authentication

Set up an authentication model which allows all passwords. Edit `/etc/origin/master/master-config.yaml` on the OpenShift master and change the value of **DenyAllPasswordIdentityProvider** to **AllowAllPasswordIdentityProvider**. Then restart the OpenShift master.

1. Now that the authentication model has been setup, login as a user, for example admin/admin:

```
# oc login openshift master e.g. https://1.1.1.1:8443 --
username=admin --password=admin
```

2. Grant the admin user account the **cluster-admin** role.

```
# oc login -u system:admin -n default
Logged into "https:// <<openshift_master_fqdn>>:8443" as
"system:admin" using existing credentials.
```

You have access to the following projects and can switch between them with 'oc project <projectname>':

```
*default
glusterfs
infra-storage
kube-public
kube-system
management-infra
openshift
openshift-infra
openshift-logging
openshift-node
openshift-sdn
openshift-web-console
```

Using project "default".

```
# oc adm policy add-cluster-role-to-user cluster-admin admin
cluster role "cluster-admin" added: "admin"
```

For more information on authentication methods, see https://access.redhat.com/documentation/en-us/openshift_container_platform/3.10/html-single/configuring_clusters/#identity-providers-configuring.

CHAPTER 7. GLUSTER BLOCK STORAGE AS BACKEND FOR LOGGING AND METRICS

Following section guides to configure Gluster Block Storage as the backend storage for logging and metrics



NOTE

Block volume expansion is not supported in CNS 3.6. Administrators are required to do proper capacity planning while using Gluster Block as backend storage when using dynamic provisioning.

7.1. PREREQUISITES

Before setting gluster block storage as the backend for logging or metrics, check if the following prerequisites are met:

- In the storageclass file, check if the default storage class is set to the storage class of gluster block. For example:

```
# oc get storageclass
NAME                                TYPE
gluster-block                      gluster.org/glusterblock
```

- If the default is not set to **gluster-block** (or any other name that you have provided) then execute the following command. For example:

```
# oc patch storageclass gluster-block -p '{"metadata":
{"annotations":{"storageclass.kubernetes.io/is-default-
class":"true"}}}'
```

- Execute the following command to verify:

```
oc get storageclass
NAME                                TYPE
gluster-block (default)           gluster.org/glusterblock
```

7.2. ENABLING GLUSTER BLOCK STORAGE AS BACKEND FOR LOGGING

Follow the tasks mentioned below to enable Gluster Block Storage as backend for logging:

1. To enable logging in Openshift Container platform, see https://access.redhat.com/documentation/en-us/openshift_container_platform/3.10/html-single/configuring_clusters/#install-config-aggregate-logging
2. The **openshift_logging_es_pvc_dynamic** ansible variable has to be set to true.

```
[OSEv3:vars] openshift_logging_es_pvc_dynamic=true
```

For example, a sample set of variables for `openshift_logging_` are listed below.


```

openshift_logging_install_logging=true
openshift_logging_es_pvc_dynamic=true
openshift_logging_kibana_nodeselector={"node-
role.kubernetes.io/infra": "true"}
openshift_logging_curator_nodeselector={"node-
role.kubernetes.io/infra": "true"}
openshift_logging_es_nodeselector={"node-role.kubernetes.io/infra":
"true"}
openshift_logging_es_pvc_size=10Gi
openshift_logging_es_pvc_storage_class_name="glusterfs-registry-
block"

```

3. Run the Ansible playbook. For more information, see https://access.redhat.com/documentation/en-us/openshift_container_platform/3.10/html-single/configuring_clusters/#install-config-aggregate-logging
4. To verify, execute the following command:

```
# oc get pods -n openshift-logging
```



NOTE

For more information regarding logging storage considerations, see https://access.redhat.com/documentation/en-us/openshift_container_platform/3.10/html-single/configuring_clusters/#install-config-aggregate-logging-sizing-guidelines-storage.

7.3. ENABLING GLUSTER BLOCK STORAGE AS BACKEND FOR METRICS

Follow the tasks mentioned below to enable Gluster Block Storage as backend for metrics



NOTE

By default, since Container Native Storage performs three-way replication, data will be available to the restarted node from anywhere in the cluster. As a result, it is recommended that Cassandra-level replication is turned off to avoid capacity overhead

1. To enable metrics in Openshift Container platform, see https://access.redhat.com/documentation/en-us/openshift_container_platform/3.10/html-single/configuring_clusters/#install-config-cluster-metrics
2. The **openshift_metrics_cassandra_storage_type** ansible variable should be set to **dynamic**:

```
[OSEv3:vars]openshift_metrics_cassandra_storage_type=dynamic
```

For example, a sample set of variables for `openshift_metrics_` are listed below.

```

openshift_metrics_install_metrics=true
openshift_metrics_storage_kind=dynamic
openshift_metrics_hawkular_nodeselector={"node-
role.kubernetes.io/infra": "true"}

```

```
openshift_metrics_cassandra_nodeselector={"node-
role.kubernetes.io/infra": "true"}
openshift_metrics_heapster_nodeselector={"node-
role.kubernetes.io/infra": "true"}
openshift_metrics_storage_volume_size=10Gi
openshift_metrics_cassandra_pvc_storage_class_name="glusterfs-
registry-block"
```

3. Run the Ansible playbook. For more information, see https://access.redhat.com/documentation/en-us/openshift_container_platform/3.10/html-single/configuring_clusters/#install-config-cluster-metrics.
4. To verify, execute the following command:

```
# oc get pods --n openshift-infra
```

It should list the following pods running:

```
heapster-cassandra
heapster-metrics
hawkular-&*9
```



NOTE

For more information regarding metrics storage considerations, see https://access.redhat.com/documentation/en-us/openshift_container_platform/3.10/html-single/configuring_clusters/#metrics-data-storage.

7.4. VERIFYING IF GLUSTER BLOCK IS SETUP AS BACKEND

Execute the following commands to verify if gluster block is setup as the backend for logging and metrics:

1. To get an overview of the infrastructure, execute the following command:

```
# oc get pods -n logging -o jsonpath='{range
.items[*].status.containerStatuses[*]}{"Name: "}{.name}{"\n  "}
{"Image: "}{.image}{"\n"}{"  State: "}{.state}{"\n"}{end}'
```

2. To get the details of all the persistent volume claims, execute the following command:

```
# oc get pvc
```

3. To get the details of the pvc, execute the following command:

```
# oc describe pvc <claim_name>
```

Verify the volume is mountable and that permissions allow read/write. Also, PVC claim name should match the dynamically provisioned gluster block storage class.

For more information, see https://access.redhat.com/documentation/en-us/openshift_container_platform/3.10/html-single/configuring_clusters/#install-config-aggregate-logging-sizing.

PART III. SECURITY

CHAPTER 8. ENABLING ENCRYPTION

Red Hat Gluster Storage supports network encryption using TLS/SSL. Red Hat Gluster Storage uses TLS/SSL for authentication and authorization, in place of the home grown authentication framework used for normal connections. Red Hat Gluster Storage supports the following encryption types:

- I/O encryption - encryption of the I/O connections between the Red Hat Gluster Storage clients and servers.
- Management encryption - encryption of the management (glusterd) connections within a trusted storage pool.

8.1. PREREQUISITES

To enable encryption it is necessary to have 3 certificates per node (glusterfs.key, glusterfs.pem and glusterfs.ca). For more information about the steps to be performed as prerequisites, see https://access.redhat.com/documentation/en-us/red_hat_gluster_storage/3.4/html-single/administration_guide/#chap-Network_Encryption-Prereqs.

Ensure to enable encryption while registering the storageclass file using the volumeoptions parameter. For more information on registering a storageclass file for File storage, see https://access.qa.redhat.com/documentation/en-us/red_hat_openshift_container_storage/3.10/html-single/operations_guide/#chap-Documentation-Red_Hat_Gluster_Storage_Container_Native_with_OpenShift_Platform-OpenShift_Creating_Persistent_Volumes-Dynamic_Prov.



NOTE

- Ensure to perform the steps on all the OpenShift nodes except master.
- All the Red Hat Gluster Storage volumes are mounted on the OpenShift nodes and then bind mounted to the application pods. Hence, it is not required to perform any encryption related operations specifically on the application pods.

8.2. ENABLING ENCRYPTION FOR A NEW RED HAT OPENSIFT CONTAINER STORAGE SETUP

You can configure network encryption for a new Red Hat Openshift Container Storage setup for both I/O encryption and management encryption.

8.2.1. Enabling Management Encryption

Though Red Hat Gluster Storage can be configured only for I/O encryption without using management encryption, it is recommended to have management encryption. If you want to enable SSL only on the I/O path, skip this section and proceed with [Section 8.2.2, “Enabling I/O encryption for a Volume”](#).

On the server

Perform the following on all the server, ie, the OpenShift nodes on which Red Hat Gluster Storage pods are running.

1. Create the /var/lib/glusterd/secure-access file.

```
# touch /var/lib/glusterd/secure-access
```

On the clients

Perform the following on the clients, that is, on all the remaining OpenShift nodes on which Red Hat Gluster Storage is not running.

1. Create the `/var/lib/glusterd/secure-access` file.

```
# touch /var/lib/glusterd/secure-access
```



NOTE

All the Red Hat Gluster Storage volumes are mounted on the OpenShift nodes and then bind mounted to the application pods. Hence, it is not required to perform any encryption related operations specifically on the application pods.

After running the commands on the server and clients, deploy Red Hat OpenShift Container Storage. For more information, see https://access.redhat.com/documentation/en-us/red_hat_openshift_container_storage/3.10/html-single/deployment_guide/#chap-Documentation-Red_Hat_Gluster_Storage_Container_Native_with_OpenShift_Platform-Setting_the_environment-Deploy_CNS

8.2.2. Enabling I/O encryption for a Volume

Enable the I/O encryption between the servers and clients:



NOTE

The servers are the OpenShift nodes on which Red Hat Gluster Storage pods are running.

The clients are the remaining OpenShift nodes on which Red Hat Gluster Storage is not running.

1. Ensure Red Hat OpenShift Container Storage is deployed before proceeding with further steps. For more information see, https://access.redhat.com/documentation/en-us/red_hat_openshift_container_storage/3.10/html-single/deployment_guide/#chap-Documentation-Red_Hat_Gluster_Storage_Container_Native_with_OpenShift_Platform-Setting_the_environment-Deploy_CNS
2. You can either create a statically provisioned volume or a dynamically provisioned volume. For more information about static provisioning of volumes, see https://access.redhat.com/documentation/en-us/red_hat_openshift_container_storage/3.10/html-single/operations_guide/#chap-Documentation-Red_Hat_Gluster_Storage_Container_Native_with_OpenShift_Platform-OpenShift_Creating_Persistent_Volumes-Static_Prov. For more information about dynamic provisioning of volumes, see https://access.redhat.com/documentation/en-us/red_hat_openshift_container_storage/3.10/html-single/operations_guide/#chap-Documentation-Red_Hat_Gluster_Storage_Container_Native_with_OpenShift_Platform-OpenShift_Creating_Persistent_Volumes-Dynamic_Prov

**NOTE**

To enable encryption during the creation of statically provisioned volume, execute the following command:

```
# heketi-cli volume create --size=100 --gluster-volume-
options="client.ssl on","server.ssl on"
```

3. Stop the volume by executing the following command:

```
# oc rsh <gluster_pod_name> gluster volume stop VOLNAME
```

The *gluster pod name* is the name of one of the Red Hat Gluster Storage pods of the trusted storage pool to which the volume belongs.

**NOTE**

To get the VOLNAME, execute the following command:

```
# oc describe pv <pv_name>
```

For example:

```
# oc describe pv pvc-01569c5c-1ec9-11e7-a794-
005056b38171
Name:          pvc-01569c5c-1ec9-11e7-a794-005056b38171
Labels:        <none>
StorageClass:  fast
Status:        Bound
Claim:         storage-project/storage-claim68
Reclaim Policy: Delete
Access Modes:  RW0
Capacity:      1Gi
Message:
Source:
  Type:          Glusterfs (a Glusterfs mount on
the host that shares a pod's lifetime)
  EndpointsName: glusterfs-dynamic-storage-claim68
  Path:
vol_0e81e5d6e46dcbf02c11ffd9721fca28
  ReadOnly:      false
No events.
```

The VOLNAME is the value of "path" in the above output.

4. Set the list of common names of all the servers to access the volume. Ensure to include the common names of clients which will be allowed to access the volume.

```
# oc rsh <gluster_pod_name> gluster volume set VOLNAME auth.ssl-
allow 'server1,server2,server3,client1,client2,client3'
```

**NOTE**

If you set `auth.ssl-allow` option with `*` as value, any TLS authenticated clients can mount and access the volume from the application side. Hence, you set the option's value to `*` or provide common names of clients as well as the nodes in the trusted storage pool.

5. Enable the `client.ssl` and `server.ssl` options on the volume.

```
# oc rsh <gluster_pod_name> gluster volume set VOLNAME client.ssl on
# oc rsh <gluster_pod_name> gluster volume set VOLNAME server.ssl on
```

6. Start the volume.

```
# oc rsh <gluster_pod_name> gluster volume start VOLNAME
```

8.3. ENABLING ENCRYPTION FOR AN EXISTING RED HAT OPENSIFT CONTAINER STORAGE SETUP

You can configure network encryption for an existing Red Hat Openshift Container Storage Storage setup for both I/O encryption and management encryption.

8.3.1. Enabling I/O encryption for a Volume

Enable the I/O encryption between the servers and clients for a volume:

**NOTE**

The servers are the OpenShift nodes on which Red Hat Gluster Storage pods are running.

The clients are the remaining OpenShift nodes on which Red Hat Gluster Storage is not running.

1. Stop all the application pods that have the Red Hat Gluster Storage volumes.
2. Stop the volume.

```
# oc rsh <gluster_pod_name> gluster volume stop VOLNAME
```

The *gluster pod name* is the name of one of the Red Hat Gluster Storage pods of the trusted storage pool to which the volume belongs.

3. Set the list of common names for clients allowed to access the volume. Be sure to include the common names of all the servers.

```
# oc rsh <gluster_pod_name> gluster volume set VOLNAME auth.ssl-allow 'server1,server2,server3,client1,client2,client3'
```


**NOTE**

If you set `auth.ssl-allow` option with `*` as value, any TLS authenticated clients can mount and access the volume from the application side. Hence, you set the option's value to `*` or provide common names of clients as well as the nodes in the trusted storage pool.

4. Enable `client.ssl` and `server.ssl` on the volume by using the following command:

```
# oc rsh <gluster_pod_name> gluster volume set VOLNAME client.ssl on
# oc rsh <gluster_pod_name> gluster volume set VOLNAME server.ssl on
```

5. Start the volume.

```
# oc rsh <gluster_pod_name> gluster volume start VOLNAME
```

6. Start the application pods to use the I/O encrypted Red Hat Gluster Storage volumes.

8.3.2. Enabling Management Encryption

Management encryption is recommended, even though, Red Hat Gluster Storage can be configured only for I/O encryption without using management encryption. On an existing installation, with running servers and clients, schedule a downtime of volumes, applications, clients, and other end-users to enable management encryption.

You cannot currently change between unencrypted and encrypted connections dynamically. Bricks and other local services on the servers and clients do not receive notifications from `glusterd` if they are running when the switch to management encryption is made.

1. Stop all the application pods that have the Red Hat Gluster Storage volumes.
2. Stop all the volumes.

```
# oc rsh <gluster_pod_name> gluster volume stop VOLNAME
```

3. Stop the Red Hat Gluster Storage pods.

```
# oc delete daemonset glusterfs
```

4. On deletion of daemon set the pods go down. To verify if the pods are down, execute the following command:

```
# oc get pods
```

5. Create the `/var/lib/glusterd/secure-access` file on all OpenShift nodes.

```
# touch /var/lib/glusterd/secure-access
```

6. Create the Red Hat Gluster Storage daemonset by executing the following command:

**NOTE**

For Ansible deployments, the image name and the version has to be specified in the template, before executing the command.

```
# oc process glusterfs | oc create -f -
```

7. On creation of daemon set the pods are started. To verify if the pods are started, execute the following command:

```
# oc get pods
```

8. Start all the volumes.

```
# oc rsh <gluster_pod_name> gluster volume start VOLNAME
```

9. Start the application pods to use the management encrypted Red Hat Gluster Storage.

8.4. DISABLING ENCRYPTION

You can disable encryption for on Red Hat OpenShift Container Storage setup in the following two scenarios:

- Disabling I/O Encryption for a Volume
- Disabling Management Encryption

8.4.1. Disabling I/O Encryption for all the Volumes

Execute the following commands to disable the I/O encryption between the servers and clients for a volume:

**NOTE**

The servers are the OpenShift nodes on which Red Hat Gluster Storage pods are running.

The clients are the remaining OpenShift nodes on which Red Hat Gluster Storage is not running.

1. Stop all the application pods that have the Red Hat Gluster Storage volumes.
2. Stop all the volumes.

```
# oc rsh <gluster_pod_name> gluster volume stop VOLNAME
```

3. Reset all the encryption options for a volume:

```
# oc rsh <gluster_pod_name> gluster volume reset VOLNAME auth.ssl-allow
# oc rsh <gluster_pod_name> gluster volume reset VOLNAME client.ssl
```

```
# oc rsh <gluster_pod_name> gluster volume reset VOLNAME server.ssl
```

4. Delete the files that were used for network encryption using the following command on all the OpenShift nodes:

```
# rm /etc/ssl/glusterfs.pem /etc/ssl/glusterfs.key  
/etc/ssl/glusterfs.ca
```



NOTE

Deleting these files in a setup where management encryption is enabled will result in glusterd failing on all gluster pods and hence should be avoided.

5. Stop the Red Hat Gluster Storage pods.

```
# oc delete daemonset glusterfs
```

6. On deletion of daemon set the pods go down. To verify if the pods are down, execute the following command:

```
# oc get pods
```

7. Create the Red Hat Gluster Storage daemonset by executing the following command:



NOTE

For Ansible deployments, the image name and the version has to be specified in the template, before executing the command.

```
# oc process glusterfs | oc create -f -
```

8. On creation of daemon set the pods are started. To verify if the pods are started, execute the following command:

```
# oc get pods
```

9. Start the volume.

```
# oc rsh <gluster_pod_name> gluster volume start VOLNAME
```

10. Start the application pods to use the I/O encrypted Red Hat Gluster Storage volumes.

8.4.2. Disabling Management Encryption

You cannot currently change between unencrypted and encrypted connections dynamically. Bricks and other local services on the servers and clients do not receive notifications from glusterd if they are running when the switch to management encryption is made.

Execute the following commands to disable the management encryption

1. Stop all the application pods that have the Red Hat Gluster Storage volumes.

2. Stop all the volumes.

```
# oc rsh <gluster_pod_name> gluster volume stop VOLNAME
```

3. Stop the Red Hat Gluster Storage pods.

```
# oc delete daemonset glusterfs
```

4. On deletion of daemon set the pods go down. To verify if the pods are down, execute the following command:

```
# oc get pods
```

5. Delete the `/var/lib/glusterd/secure-access` file on all OpenShift nodes to disable management encryption.

```
# rm /var/lib/glusterd/secure-access
```

6. Delete the files that were used for network encryption using the following command on all the OpenShift nodes:

```
# rm /etc/ssl/glusterfs.pem /etc/ssl/glusterfs.key  
/etc/ssl/glusterfs.ca
```

7. Create the Red Hat Gluster Storage daemonset by executing the following command:

**NOTE**

For Ansible deployments, the image name and the version has to be specified in the template, before executing the command.

```
# oc process glusterfs | oc create -f -
```

8. On creation of daemon set the pods are started. To verify if the pods are started, execute the following command:

```
# oc get pods
```

9. Start all the volumes.

```
# oc rsh <gluster_pod_name> gluster volume start VOLNAME
```

10. Start the application pods to use the management encrypted Red Hat Gluster Storage.

PART IV. MIGRATION

CHAPTER 9. UPDATING THE REGISTRY WITH RED HAT OPENSIFT CONTAINER STORAGE AS THE STORAGE BACK-END

OpenShift Container Platform provides an integrated registry with storage using an NFS-backed persistent volume that is automatically setup. Red Hat Openshift Container Storage allows you to replace this with a Gluster persistent volume for registry storage. This provides increased reliability, scalability and failover.

For additional information about OpenShift Container Platform and the docker-registry, see https://access.redhat.com/documentation/en-us/openshift_container_platform/3.10/html/configuring_clusters/setting-up-the-registry.

9.1. VALIDATING THE OPENSIFT CONTAINER PLATFORM REGISTRY DEPLOYMENT

To verify that the registry is properly deployed, execute the following commands:

1. On the master or client, execute the following command to login as the cluster admin user:

```
# oc login
```

For example:

```
# oc login

Authentication required for https://master.example.com:8443
(openshift)
Username: <cluster-admin-user>
Password: <password>
Login successful.

You have access to the following projects and can switch between
them with 'oc project <projectname>':

* default
  management-infra
  openshift
  openshift-infra

Using project "default".
```

If you are not automatically logged into project default, then switch to it by executing the following command:

```
# oc project default
```

2. To verify that the pod is created, execute the following command:

```
# oc get pods
```

For example:

```
# oc get pods
NAME                                READY    STATUS    RESTARTS   AGE
docker-registry-2-mbu0u            1/1      Running   4           6d
docker-registry-2-spw0o            1/1      Running   3           6d
registry-console-1-rblwo           1/1      Running   3           6d
```

3. To verify that the endpoints are created, execute the following command:

```
# oc get endpoints
```

For example:

```
# oc get endpoints
NAME                                ENDPOINTS
AGE
docker-registry                    10.128.0.15:5000,10.129.0.9:5000
7d
kubernetes                         192.168.234.143:8443,192.168.234.143:8053,192.168.234.143:8053
7d
registry-console                   10.128.0.17:9090
7d
router                             192.168.234.144:443,192.168.234.145:443,192.168.234.144:1936 + 3
more... 7d
```

4. To verify that the persistent volume is created, execute the following command:

```
# oc get pv
NAME      CAPACITY   ACCESSMODES   RECLAIMPOLICY   STATUS   CLAIM
REASON    AGE
registry-volume 5Gi        RWX           Retain          Bound    default/registry-claim
7d
```

5. To obtain the details of the persistent volume that was created for the NFS registry, execute the following command:

```
# oc describe pv registry-volume
Name:          registry-volume
Labels:        <none>
StorageClass:
Status:        Bound
Claim:         default/registry-claim
Reclaim Policy: Retain
Access Modes:  RWX
Capacity:      5Gi
Message:
Source:
  Type:        NFS (an NFS mount that lasts the lifetime of a pod)
  Server:      cns30.rh73
  Path:        /exports/registry
  ReadOnly:    false
No events.
```

9.2. CONVERTING THE OPENSIFT CONTAINER PLATFORM REGISTRY WITH RED HAT OPENSIFT CONTAINER STORAGE

This section provides the steps to create a Red Hat Gluster Storage volume and use it to provide storage for the integrated registry.

Setting up a Red Hat Gluster Storage Persistent Volume

Execute the following commands to create a Red Hat Gluster Storage volume to store the registry data and create a persistent volume.



NOTE

The commands must be executed in the **default** project.

1. Login to the **default** project:

```
# oc project default
```

For example:

```
# oc project default
Now using project "default" on server "https://cns30.rh73:8443"
```

2. Execute the following command to create the **gluster-registry-endpoints.yaml** file:

```
oc get endpoints <heketi-db-storage-endpoint-name> -o yaml --
namespace=<project-name> > gluster-registry-endpoints.yaml
```



NOTE

You must create an endpoint for each project from which you want to utilize the Red Hat Gluster Storage registry. Hence, you will have a service and an endpoint in both the **default** project and the new project (**storage-project**) created in earlier steps.

3. Edit the **gluster-registry-endpoints.yaml** file. Change the name to gluster-registry-endpoints and remove all the other metadata, leaving everything else the same.

```
# cat gluster-registry-endpoints.yaml
apiVersion: v1
kind: Endpoints
metadata:
  name: gluster-registry-endpoints
subsets:
- addresses:
  - ip: 192.168.124.114
  - ip: 192.168.124.52
  - ip: 192.168.124.83
  ports:
  - port: 1
    protocol: TCP
```


4. Execute the following command to create the endpoint:

```
# oc create -f gluster-registry-endpoints.yaml
endpoints "gluster-registry-endpoints" created
```

5. To verify the creation of the endpoint, execute the following command:

```
# oc get endpoints
NAME                                ENDPOINTS
AGE
docker-registry                    10.129.0.8:5000,10.130.0.5:5000
28d
gluster-registry-endpoints
192.168.124.114:1,192.168.124.52:1,192.168.124.83:1
10s
kubernetes
192.168.124.250:8443,192.168.124.250:8053,192.168.124.250:8053
28d
registry-console                  10.131.0.6:9090
28d
router
192.168.124.114:443,192.168.124.83:443,192.168.124.114:1936 + 3
more...    28d
```

6. Execute the following command to create the **gluster-registry-service.yaml** file:

```
oc get services <heketi-storage-endpoint-name> -o yaml --namespace=
<project-name> > gluster-registry-service.yaml
```

7. Edit the **gluster-registry-service.yaml** file. Change the name to gluster-registry-service and remove all the other metadata. Also, remove the specific cluster IP addresses:

```
# cat gluster-registry-service.yaml
apiVersion: v1
kind: Service
metadata:
  name: gluster-registry-service
spec:
  ports:
    - port: 1
      protocol: TCP
      targetPort: 1
  sessionAffinity: None
  type: ClusterIP
status:
  loadBalancer: {}
```

8. Execute the following command to create the service:

```
# oc create -f gluster-registry-service.yaml
services "gluster-registry-service" created
```

9. Execute the following command to verify if the service are running:

```
# oc get services
NAME                                CLUSTER-IP      EXTERNAL-IP      PORT(S)
AGE
docker-registry                    172.30.197.118   <none>           5000/TCP
28d
gluster-registry-service          172.30.0.183     <none>           1/TCP
6s
kubernetes                        172.30.0.1       <none>
443/TCP, 53/UDP, 53/TCP          29d
registry-console                  172.30.146.178   <none>           9000/TCP
28d
router                            172.30.232.238   <none>
80/TCP, 443/TCP, 1936/TCP       28d
```

10. Execute the following command to obtain the fsGroup GID of the existing docker-registry pods:

```
# export GID=$(oc get po --selector="docker-registry=default" -o go-
template --template='{{printf "%.0f" ((index .items
0).spec.securityContext.fsGroup)}}')
```

11. Execute the following command to create a volume

```
# heketi-cli volume create --size=5 --name=gluster-registry-volume -
-gid=${GID}
```

12. Create the persistent volume file for the Red Hat Gluster Storage volume:

```
# cat gluster-registry-volume.yaml
kind: PersistentVolume
apiVersion: v1
metadata:
  name: gluster-registry-volume
  labels:
    glusterfs: registry-volume
spec:
  capacity:
    storage: 5Gi
  glusterfs:
    endpoints: gluster-registry-endpoints
    path: gluster-registry-volume
  accessModes:
    - ReadWriteMany
  persistentVolumeReclaimPolicy: Retain
```

13. Execute the following command to create the persistent volume:

```
# oc create -f gluster-registry-volume.yaml
```

14. Execute the following command to verify and get the details of the created persistent volume:

```
# oc get pv/gluster-registry-volume
NAME                                CAPACITY  ACCESSMODES  RECLAIMPOLICY
STATUS    CLAIM    REASON    AGE
```

gluster-registry-volume	5Gi	RWX	Retain
Available	21m		

15. Create a new persistent volume claim. Following is a sample Persistent Volume Claim that will be used to replace the existing registry-storage volume claim.

```
# cat gluster-registry-claim.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: gluster-registry-claim
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 5Gi
  selector:
    matchLabels:
      glusterfs: registry-volume
```

16. Create the persistent volume claim by executing the following command:

```
# oc create -f gluster-registry-claim.yaml
```

For example:

```
# oc create -f gluster-registry-claim.yaml
persistentvolumeclaim "gluster-registry-claim" created
```

17. Execute the following command to verify if the claim is bound:

```
# oc get pvc/gluster-registry-claim
```

For example:

```
# oc get pvc/gluster-registry-claim
NAME                                STATUS    VOLUME
CAPACITY  ACCESSMODES  AGE
gluster-registry-claim  Bound      gluster-registry-volume  5Gi
RWX                22s
```

18. Make the registry read-only by executing the following command:

```
# oc set env -n default dc/docker-registry
'REGISTRY_STORAGE_MAINTENANCE_READONLY={"enabled":true}'
```

To confirm the value is set to readonly, execute the following command:

```
# oc set env -n default dc/docker-registry --list
```

19. If you want to migrate the data from the old registry to the Red Hat Gluster Storage registry, then execute the following commands:

**NOTE**

These steps are optional.

1. Add the Red Hat Gluster Storage registry to the old registry deployment configuration (dc) by executing the following command:

```
# oc volume dc/docker-registry --add --name=gluster-registry-storage -m /gluster-registry -t pvc --claim-name=gluster-registry-claim
```

2. Save the Registry pod name by executing the following command:

```
# export REGISTRY_POD=$(oc get po --selector="docker-registry=default" -o go-template --template='{{printf "%s" ((index .items 0).metadata.name)}}')
```

3. Copy the data from the old registry directory to the Red Hat Gluster Storage registry directory by executing the following command:

```
# oc rsh $REGISTRY_POD cp -a /registry/ /gluster-registry/
```

4. Remove the Red Hat Gluster Storage registry from the old dc registry by executing the following command:

```
# oc volume dc/docker-registry --remove --name=gluster-registry-storage
```

20. Replace the existing registry-storage volume with the new gluster-registry-claim PVC:

```
# oc volume dc/docker-registry --add --name=registry-storage -t pvc --claim-name=gluster-registry-claim --overwrite
```

21. Make the registry read write by executing the following command:

```
# oc set env dc/docker-registry REGISTRY_STORAGE_MAINTENANCE_READONLY-
```

To validate if the setting is set to read write, execute the following command:

```
# oc set env -n default dc/docker-registry --list
```

For more information about accessing the registry, see https://access.redhat.com/documentation/en-us/openshift_container_platform/3.10/html/configuring_clusters/setting-up-the-registry#install-config-registry-accessing.

PART V. MONITORING

CHAPTER 10. ENABLING VOLUME METRICS

To enable volume metrics for glusterfs plugin in order to collect the statistics of GlusterFS PVs, set up Prometheus, a monitoring toolkit. You can use Prometheus to visualize metrics and alerts for OpenShift Container Platform system resources.

For more information on how to setup Prometheus, see https://access.redhat.com/documentation/en-us/openshift_container_platform/3.10/html-single/configuring_clusters/#openshift-prometheus

10.1. ENABLING VOLUME METRICS FOR FILE STORAGE AND BLOCK STORAGE

Following is the list of different metrics of the PVs that can be viewed on Prometheus:

- **kubelet_volume_stats_available_bytes**: Number of available bytes in the volume.
- **kubelet_volume_stats_capacity_bytes**: Capacity in bytes of the volume.
- **kubelet_volume_stats_inodes**: Maximum number of inodes in the volume.
- **kubelet_volume_stats_inodes_free**: Number of free inodes in the volume.
- **kubelet_volume_stats_inodes_used**: Number of used inodes in the volume.
- **kubelet_volume_stats_used_bytes**: Number of used bytes in the volume.
- **heketi_cluster_count**: Number of clusters.
- **heketi_device_brick_count**: Number of bricks on device.
- **heketi_device_count**: Number of devices on host.
- **heketi_device_free**: Amount of free space available on the device.
- **heketi_device_size**: Total size of the device.
- **heketi_device_used**: Amount of space used on the device.
- **heketi_nodes_count**: Number of nodes on the cluster.
- **heketi_up**: Verifies if heketi is running.
- **heketi_volumes_count**: Number of volumes on cluster.

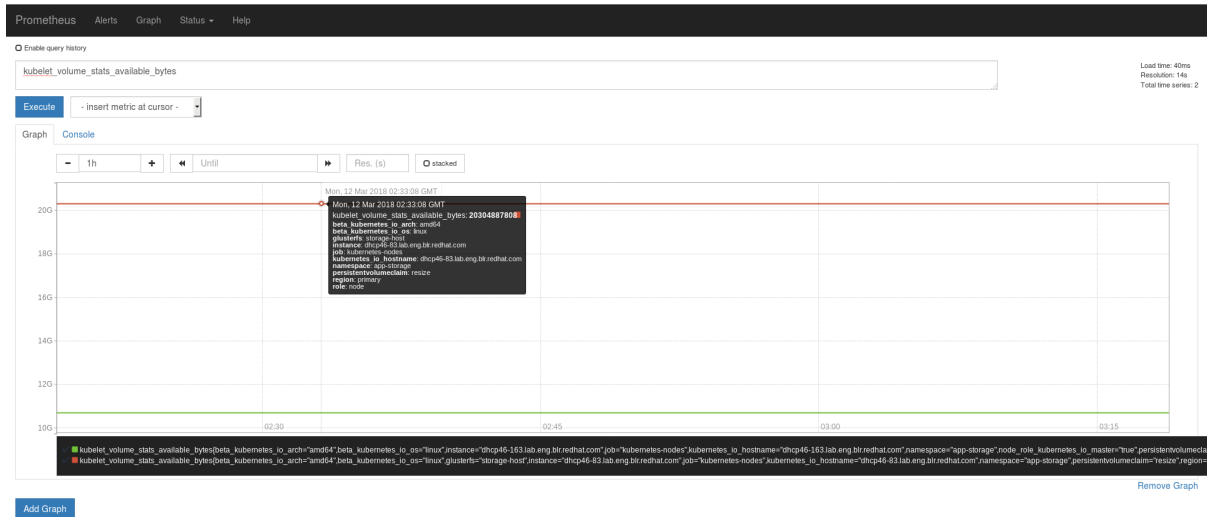
Viewing Metrics

To view any metrics:

1. Add the metrics name in **Prometheus**, and click **Execute**.
2. In the **Graph** tab, the value for the metrics for the volume is displayed as a graph.

For example, in the following image, to check the available bytes, **kubelet_volume_stats_available_bytes** metric is added to the search bar on **Prometheus**. On clicking **Execute**, the available bytes value is depicted as a graph. You can

hover the mouse on the line to get more details. (To view the image in detail, right-click and select View Image.)



Viewing Heketi Metrics

To view Heketi metrics on **Prometheus**, execute the following commands:

1. Add annotations to the **heketi-storage**.

```
# oc annotate svc heketi-storage prometheus.io/scheme=http
# oc annotate svc heketi-storage prometheus.io/scrape=true
```

```
# oc describe svc heketi-storage
Name:                heketi-storage
Namespace:            app-storage
Labels:               glusterfs=heketi-storage-service
                     heketi=storage-service
Annotations:          description=Exposes Heketi service
                     prometheus.io/scheme=http
                     prometheus.io/scrape=true
Selector:             glusterfs=heketi-storage-pod
Type:                 ClusterIP
IP:                   172.30.90.87
Port:                 heketi 8080/TCP
TargetPort:           8080/TCP
Endpoints:            172.18.14.2:8080
Session Affinity:     None
```

2. Add the **app-storage** namespace for the heketi service in the Prometheus configmap.

```
# oc get cm prometheus -o yaml -n openshift-metrics
....
- job_name: 'kubernetes-service-endpoints'

  tls_config:
  ca_file:
/var/run/secrets/kubernetes.io/serviceaccount/ca.crt
  # TODO: this should be per target
  insecure_skip_verify: true
```

```
kubernetes_sd_configs:
- role: endpoints

relabel_configs:
# only scrape infrastructure components
- source_labels: [__meta_kubernetes_namespace]
  action: keep
  regex: 'default|logging|metrics|kube-
.+|openshift|openshift-.+|app-storage'
```

3. Restart the **prometheus-0** pod to query the Heketi metrics in Prometheus.

PART VI. TROUBLESHOOT

CHAPTER 11. TROUBLESHOOTING

This chapter describes the most common troubleshooting scenarios related to Red Hat OpenShift Container Storage.

- **What to do if a Red Hat OpenShift Container Storage node Fails**

If a Red Hat OpenShift Container Storage node fails, and you want to delete it, then, disable the node before deleting it. For more information, see [Section 1.2.3, “Deleting Node”](#).

If a Red Hat OpenShift Container Storage node fails and you want to replace it, see [Section 1.2.3.3, “Replacing a Node”](#).

- **What to do if a Red Hat OpenShift Container Storage device fails**

If a Red Hat OpenShift Container Storage device fails, and you want to delete it, then, disable the device before deleting it. For more information, see [Section 1.2.2, “Deleting Device”](#).

If a Red Hat OpenShift Container Storage device fails, and you want to replace it, see [Section 1.2.2.3, “Replacing a Device”](#).

- **What to do if Red Hat OpenShift Container Storage volumes require more capacity:**

You can increase the storage capacity by either adding devices, increasing the cluster size, or adding an entirely new cluster. For more information, see [Section 1.1, “Increasing Storage Capacity”](#).

- **How to upgrade OpenShift when Red Hat OpenShift Container Storage is installed**

To upgrade OpenShift Container Platform, see https://access.redhat.com/documentation/en-us/openshift_container_platform/3.10/html/upgrading_clusters/install-config-upgrading-automated-upgrades#upgrading-to-ocp-3-10.

- **Viewing Log Files**

- **Viewing Red Hat Gluster Storage Container Logs**

Debugging information related to Red Hat Gluster Storage containers is stored on the host where the containers are started. Specifically, the logs and configuration files can be found at the following locations on the openshift nodes where the Red Hat Gluster Storage server containers run:

- `/etc/glusterfs`
- `/var/lib/glusterd`
- `/var/log/glusterfs`

- **Viewing Heketi Logs**

Debugging information related to Heketi is stored locally in the container or in the persisted volume that is provided to Heketi container.

You can obtain logs for Heketi by running the `docker logs container-id` command on the openshift node where the container is being run.

- **Heketi command returns with no error or empty error**

Sometimes, running `heketi-cli` command returns with no error or empty error like **Error**. It is mostly due to `heketi` server not properly configured. You must first ping to validate that the `Heketi` server is available and later verify with a `curl` command and **/hello endpoint**.

```
# curl http://deploy-heketi-storage-
project.cloudapps.mystorage.com/hello
```

- **Heketi reports an error while loading the topology file**

Running `heketi-cli` reports : Error "Unable to open topology file" error while loading the topology file. This could be due to the use of old syntax of single hyphen (-) as a prefix for JSON option. You must use the new syntax of double hyphens and reload the topology file.

- **cURL command to `heketi` server fails or does not respond**

If the router or `heketi` is not configured properly, error messages from the `heketi` may not be clear. To troubleshoot, ping the `heketi` service using the endpoint and also using the IP address. If ping by the IP address succeeds and ping by the endpoint fails, it indicates a router configuration error.

After the router is setup properly, run a simple `curl` command like the following:

```
# curl http://deploy-heketi-storage-
project.cloudapps.mystorage.com/hello
```

If `heketi` is configured correctly, a welcome message from `heketi` is displayed. If not, check the `heketi` configuration.

- **Heketi fails to start when Red Hat Gluster Storage volume is used to store `heketi.db` file**

Sometimes `Heketi` fails to start when Red Hat Gluster Storage volume is used to store `heketi.db` and reports the following error:

```
[heketi] INFO 2016/06/23 08:33:47 Loaded kubernetes executor
[heketi] ERROR 2016/06/23 08:33:47
/src/github.com/heketi/heketi/apps/glusterfs/app.go:149: write
/var/lib/heketi/heketi.db: read-only file system
ERROR: Unable to start application
```

The read-only file system error as shown above could be seen while using a Red Hat Gluster Storage volume as backend. This could be when the quorum is lost for the Red Hat Gluster Storage volume. In a replica-3 volume, this would be seen if 2 of the 3 bricks are down. You must ensure the quorum is met for `heketi` gluster volume and it is able to write to `heketi.db` file again.

Even if you see a different error, it is a recommended practice to check if the Red Hat Gluster Storage volume serving `heketi.db` file is available or not. Access deny to `heketi.db` file is the most common reason for it to not start.

CHAPTER 12. CLIENT CONFIGURATION USING PORT FORWARDING

If a router is not available, you may be able to set up port forwarding so that `heketi-cli` can communicate with the Heketi service. Execute the following commands for port forwarding:

1. Obtain the Heketi service pod name by running the following command:

```
# oc get pods
```

2. To forward the port on your local system to the pod, execute the following command on another terminal of your local system:

```
# oc port-forward <heketi pod name> 8080:8080
```

3. On the original terminal execute the following command to test the communication with the server:

```
# curl http://localhost:8080/hello
```

This will forward the local port 8080 to the pod port 8080.

4. Setup the Heketi server environment variable by running the following command:

```
# export HEKETI_CLI_SERVER=http://localhost:8080
```

5. Get information from Heketi by running the following command:

```
# heketi-cli topology info
```

APPENDIX A. REVISION HISTORY

Revision 1.0-02**Wed Sep 12 2018****Bhavana Mohan**

Publishing for Red Hat OpenShift Container Storage 3.10

Revision 1.0-01**Tue Sep 11 2018****Bhavana Mohan**

Rebranded the product from Container-Native Storage to Red Hat OpenShift Container Storage. The terms describing the method of deployment have also been changed to Converged mode (formerly referred to as 'Container Native Storage' or 'CNS') and Independent mode (formerly referred to as 'Container Ready Storage' or 'CRS')

Created two separate guides; one for Deployment and upgrade and another for administrative tasks.

Documented details about enhanced OCS monitoring and configuration visibility using the Prometheus framework

Documented detailed procedure to reclaim persistent volume

Documented detailed procedure to replace a block on block storage

INDEX