Container-Native Storage 3.9

Container-Native Storage for OpenShift Container Platform

Deploying Container-Native Storage for OpenShift Container Platform 3.9

Edition 1

Last Updated: 2018-04-26

Container-Native Storage 3.9 Container-Native Storage for OpenShift Container Platform

Deploying Container-Native Storage for OpenShift Container Platform 3.9
Edition 1

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

Legal Notice

Abstract

This guide describes the prerequisites and provides step-by-step instructions to deploy Container-Native Storage with OpenShift Platform.

Table of Contents

CHAPTER 1. INTRODUCTION TO CONTAINERIZED RED HAT GLUSTER STORAGE

This guide provides step-by-step instructions to deploy Containerized Red Hat Gluster Storage. The deployment addresses the use-case where applications require both shared file storage and the flexibility of a converged infrastructure with compute and storage instances being scheduled and run from the same set of hardware.

Containerized Red Hat Gluster Storage Solutions

The following table lists the Containerized Red Hat Gluster Storage solutions, a brief description, and the links to the documentation for more information about the solution.

Table 1.1. Containerized Red Hat Gluster Storage Solutions

Solution	Description	Documentation
Container-Native Storage (CNS)	This solution addresses the use-case where applications require both shared file storage and the flexibility of a converged infrastructure with compute and storage instances being scheduled and run from the same set of hardware.	For information on deploying CNS, see Chapter 2, *Container-Native Storage for OpenShift Container Platform* in this guide.
Container Ready Storage (CRS) with Heketi	This solution addresses the use-case where a dedicated Gluster cluster is available external to the OpenShift Origin cluster, and you provision storage from the Gluster cluster. In this mode, Heketi also runs outside the cluster and and can be co-located with a Red Hat Gluster Storage node.	For information on configuring CRS with Heketi, see Complete Example of Dynamic Provisioning Using Dedicated GlusterFS.
Container Ready Storage (CRS) without Heketi	This solution uses your OpenShift Container Platform cluster (without Heketi) to provision Red Hat Gluster Storage volumes (from a dedicated Red Hat Gluster Storage cluster) as persistent storage for containerized applications.	For information on creating OpenShift Container Platform cluster with persistent storage using Red Hat Gluster Storage, see Persistent Storage Using GlusterFS .

CHAPTER 2. CONTAINER-NATIVE STORAGE FOR OPENSHIFT CONTAINER PLATFORM

This deployment delivers a hyper-converged solution, where the storage containers that host Red Hat Gluster Storage co-reside with the compute containers and serve out storage from the hosts that have local or direct attached storage to the compute containers. This solution integrates Red Hat Gluster Storage deployment and management with OpenShift services. As a result, persistent storage is delivered within an OpenShift pod that provides both compute and file storage.

Container-Native Storage for OpenShift Container Platform is built around three key technologies:

- OpenShift provides the platform as a service (PaaS) infrastructure based on Kubernetes container management. Basic OpenShift architecture is built around multiple master systems where each system contains a set of nodes.

- Red Hat Gluster Storage provides the containerized distributed storage based on Red Hat Gluster Storage 3.3 container. Each Red Hat Gluster Storage volume is composed of a collection of bricks, where each brick is the combination of a node and an export directory.

- Heketi provides the Red Hat Gluster Storage volume life cycle management. It creates the Red Hat Gluster Storage volumes dynamically and supports multiple Red Hat Gluster Storage clusters.

The following list provides the administrators a solution workflow. The administrators can:

- Create multiple persistent volumes (PV) and register these volumes with OpenShift.

- Developers then submit a persistent volume claim (PVC).

- A PV is identified and selected from a pool of available PVs and bound to the PVC.

- The OpenShift pod then uses the PV for persistent storage.

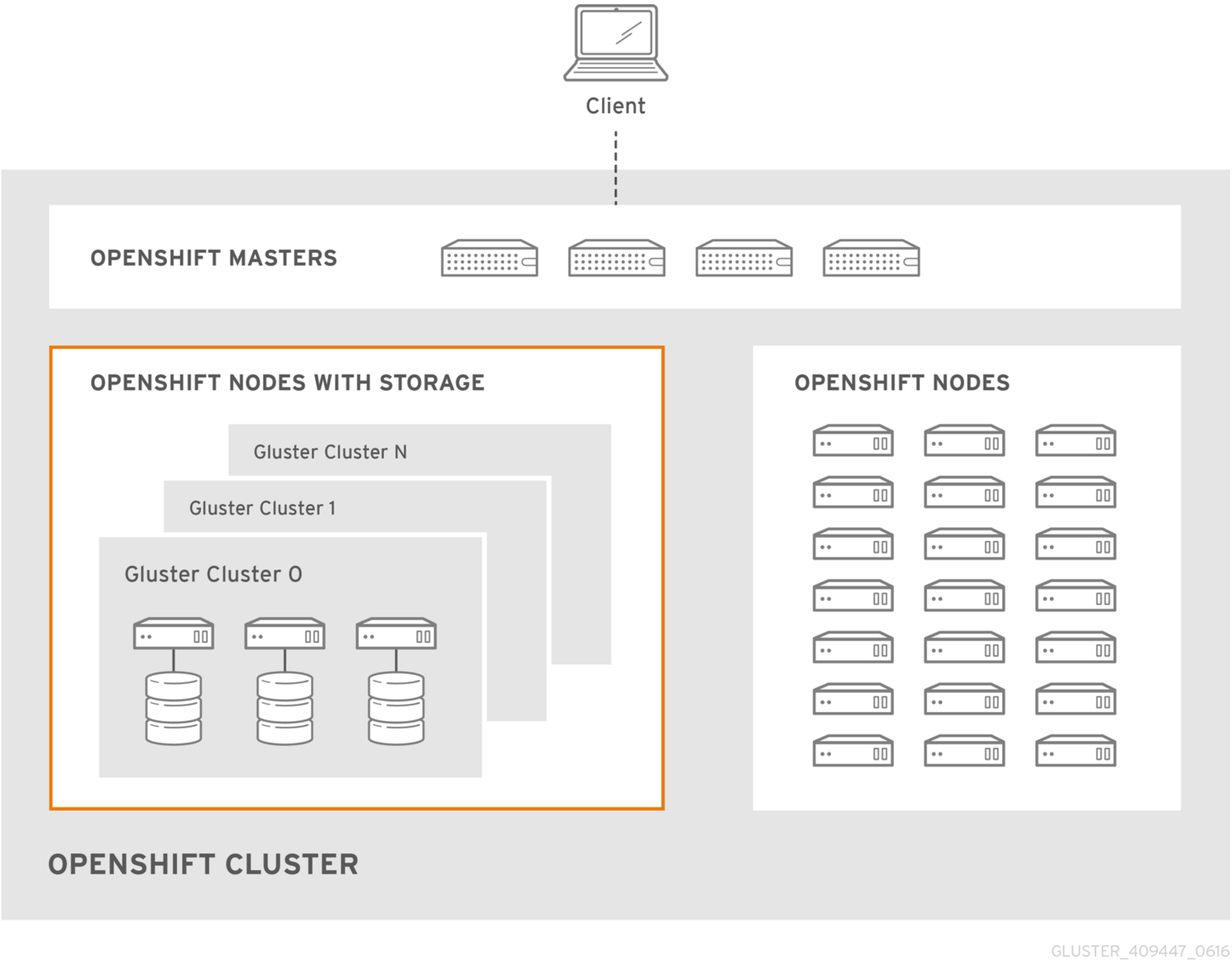


Figure 2.1. Architecture - Container-Native Storage for OpenShift Container Platform

CHAPTER 3. CONTAINER-READY STORAGE FOR OPENSHIFT CONTAINER PLATFORM

Container-Ready Storage is deployed as a stand-alone Red Hat Gluster Storage cluster that provides persistent storage to containers, unlike Container-Native Storage, which is deployed on top of an OpenShift Cluster.

Container-Ready Storage provides the same storage functionality to OpenShift Container Platform as Container-Native Storage. Container-Ready Storage provides dynamic provisioned storage, statically provisioned storage, RWO support, and RWX support. Further, it provides full support for OpenShift Container Platform infrastructure services like logging, metrics, and registry services. Being stand-alone of OpenShift Container Platform, Container-Ready Storage does have an advantage regarding providing additional Red Hat Gluster Storage data services functionality to what is supported by OpenShift, such as, Snapshot, Geo Replication, and Nagios Monitoring.

For users of persistent storage, the deployment modes are completely transparent. Administrators will see variation in how they set the system up, manage, and scale. In Container-Ready Storage, storage is managed like Red Hat Gluster Storage.

Following are some of the key drivers of choosing Container-Ready Storage mode of deployment:

- OpenShift Container Platform administrators might not want to manage storage. Container-Ready Storage separates storage management from container management.

- Leverage legacy storage (SAN, Arrays, Old filers): Customers often have storage arrays from traditional storage vendors that have either limited or no support for OpenShift. Container-Ready Storage mode allows users to leverage existing legacy storage for OpenShift Containers.

- Cost effective: In environments where costs related to new infrastructure is a challenge, they can re-purpose their existing storage arrays to back OpenShift under Container-Ready Storage. Container-Ready Storage is perfect for such situations where one can run Red Hat Gluster Storage inside a VM and serve out LUNs or disks from these storage arrays to OpenShift offering all of the features that the OpenShift storage subsystem has to offer including dynamic provisioning. This is a very useful solution in those environments with potential infrastructure additions.

Container-Ready Storage may have Heketi, and other provisioners (components of Container-Ready Storage) deployed on top of OpenShift Cluster nodes. With Container-Native Storage 3.9, Red Hat recommends Heketi be deployed on OpenShift Cluster. Heketi is a service endpoint for automated Red Hat Gluster Storage volume provisioning, where requests for allocation of Red Hat Gluster Storage volumes to back OpenShift PVs land from kubernetes. Heketi manages allocation and de-allocation of Red Hat Gluster Storage volumes dynamically.

CHAPTER 4. INSTALL AND UPGRADE WORKFLOW: WHAT TASKS DO I NEED TO COMPLETE?

The recommended, best practice method for deploying Container-Native Storage 3.9 and Container-Ready Storage 3.9 is to deploy with the Openshift Container Platform 3.9 Advanced Installer, as documented in the "24.3. Persistent Storage Using Red Hat Gluster Storage" procedure in the OpenShift Container Platform 3.9 Installation and Configuration Guide:

- Persistent Storage Using Red Hat Gluster Storage.

The following content in this "Install and Upgrade Workflow" chapter lists the workflow for different use cases for Installing or Upgrading a Container-Native Storage or Container-Ready Storage environment with the cns-deploy tool.

4.1. (INSTALL) EXISTING ENVIRONMENT: OPENSHIFT CONTAINER PLATFORM AND CONTAINER-NATIVE STORAGE ARE NOT INSTALLED

4.1.1. Customer Objective

Install OpenShift Container Platform 3.9 and Container-Native Storage 3.9.

4.1.2. Prerequisites

- Install the registry with NFS backend when installing OpenShift Container Platform.
- Do not install Logging and Metrics when installing OpenShift Container Platform.
- Red Hat Gluster Storage Requirements
- Planning Guidelines

4.1.3. Required Installation Tasks

1. OpenShift Container Platform 3.9 Quick Installation or OpenShift Container Platform 3.9 Advanced Installation
2. Container-Native Storage Environment Requirements on RHEL 7 or Container-Native Storage Environment Requirements on RHEL Atomic Host
3. Red Hat OpenShift Container Platform Requirements
4. Setting up Container-Native Storage
5. Setting up the Environment and Deploying Container-Native Storage
6. Migrate registry back-end to Gluster: Migrating Registry
7. To use Block Storage: Block Storage
8. To set Gluster-block as backend for Logging and Metrics: Logging and Metrics
9. To use File Storage: File Storage

4.2. (INSTALL) EXISTING ENVIRONMENT: OPENSHIFT CONTAINER PLATFORM AND CONTAINER-READY STORAGE ARE NOT INSTALLED

4.2.1. Customer Objective

Install OpenShift Container Platform 3.9 and Container- Ready Storage.

4.2.2. Prerequisites

- Install the registry with NFS backend when installing OpenShift Container Platform.
- Do not install Logging and Metrics when installing OpenShift Container Platform.
- Planning Guidelines

4.2.3. Required Installation Tasks

1. OpenShift Container Platform 3.9 Quick Installation or OpenShift Container Platform 3.9 Advanced Installation
2. Container-Ready Storage Environment Requirements on RHEL 7 or Container-Ready Storage Environment Requirements on RHEL Atomic Host
3. Container-Ready Storage Requirements
4. Setting up Container-Ready Storage
5. Setting up the Environment and Deploying Container-Ready Storage
6. Migrate registry backend to Gluster: Migrating Registry
7. To use Block Storage: Block Storage
8. To set Gluster-block as backend for logging and metrics: Logging and Metrics
9. To use File Storage: File Storage

4.3. (INSTALL) EXISTING ENVIRONMENT: OPENSHIFT CONTAINER PLATFORM 3.9 IS INSTALLED AND CONTAINER-NATIVE STORAGE 3.9 IS NOT INSTALLED

4.3.1. Customer Objective

Install Container-Native Storage 3.9.

4.3.2. Prerequisites

- Red Hat Gluster Storage Requirements
- Planning Guidelines

4.3.3. Required Installation Tasks

1. If the registry was not setup during OpenShift Container Platform 3.9 installation, make sure to follow the advanced installation of OpenShift Container Platform 3.9 to setup registry with NFS as the backend. The ansible variable to be set is **`openshift_hosted_registry_storage_kind=nfs`** : Advanced Installation

   Refer section 2.6.3.9: Configuring the OpenShift Container Registry.

2. Container-Native Storage Environment Requirements on RHEL 7 or Container-Native Storage Environment Requirements on RHEL Atomic Host
3. Red Hat OpenShift Container Platform Requirements
4. Setting up Container-Native Storage
5. Setting up the Environment
6. Migrate registry backend to Gluster: Migrating Registry
7. To use Block Storage: Block Storage
8. To set Gluster-block as backend for logging and metrics: Logging and Metrics
9. To use File Storage: File Storage

4.4. (INSTALL) EXISTING ENVIRONMENT: OPENSHIFT CONTAINER PLATFORM 3.9 IS INSTALLED AND CONTAINER-READY STORAGE IS NOT INSTALLED

4.4.1. Customer Objective

Install Container-Ready Storage.

4.4.2. Prerequisites

- Planning Guidelines

4.4.3. Required Installation Tasks

1. If the registry was not set up during OpenShift Container Platform installation, make sure to follow the advanced installation of OpenShift Container Platform to setup registry with NFS as the backend. The ansible variable to be set is **`openshift_hosted_registry_storage_kind=nfs`** : Advanced Installation
2. Container-Ready Storage Environment Requirements on RHEL 7 or Container-Ready Storage Environment Requirements on RHEL Atomic Host
3. Container-Ready Storage Requirements
4. Setting up Container-Ready Storage
5. Setting up the Environment
6. Migrating the registry backend to gluster: Migrating Registry
7. To use block storage: Block Storage

8. To set Gluster Block as back-end for Logging and Metrics: Logging and Metrics

9. To use File Storage: File Storage

4.5. (UPGRADE) EXISTING ENVIRONMENT: OPENSHIFT CONTAINER PLATFORM 3.7 OR 3.9 IS INSTALLED AND CONTAINER-NATIVE STORAGE IS INSTALLED

4.5.1. Customer Objective

These steps are applicable for 3 scenarios:

- OpenShift Container Platform 3.7 or 3.9 is installed and Container-Native Storage 3.6 is installed with Advanced Installer and Registry

- OpenShift Container Platform 3.7 or 3.9 is installed and Container-Native Storage 3.9 is installed with Advanced Installer and Registry

- OpenShift Container Platform 3.7 or 3.9 is installed and Container-Native Storage 3.6 is installed using cns-deploy tool.

4.5.2. Required Upgrade Tasks

1. If the registry was not set up during OpenShift Container Platform installation, make sure to follow the advanced installation of OpenShift Container Platform to setup registry with NFS as the backend. The ansible variable to be set is `openshift_hosted_registry_storage_kind=nfs` : Advanced Installation

2. Upgrading Container-Native Storage.

3. Migrating the registry backend to gluster: Migrating Registry

4. To use block storage: Block Storage

5. To set Gluster Block as back-end for Logging and Metrics: Logging and Metrics

6. To use File Storage: File Storage

4.6. (UPGRADE) EXISTING ENVIRONMENT: OPENSHIFT CONTAINER PLATFORM 3.7 OR 3.9 IS INSTALLED AND CONTAINER-READY STORAGE IS INSTALLED WITH ADVANCED INSTALLER AND REGISTRY

4.6.1. Customer Objective

Upgrade Container-Ready Storage with all functions.

4.6.2. Required Upgrade Tasks

1. If the registry was not set up during OpenShift Container Platform installation, make sure to follow the advanced installation of OpenShift Container Platform to setup registry with NFS as the backend. The ansible variable to be set is

`openshift_hosted_registry_storage_kind=nfs` : Advanced Installation

2. Upgrading Container-Ready Storage.

NOTE

Execute only the steps that are relevant to your environment.

3. Migrating the registry backend to gluster: Migrating Registry

4. To use block storage: Block Storage

5. To set Gluster Block as back-end for Logging and Metrics: Logging and Metrics

6. To use File Storage: File Storage

CHAPTER 5. SUPPORT REQUIREMENTS

This chapter describes and lists the various prerequisites to set up Red Hat Gluster Storage Container Native with OpenShift Container Platform.

5.1. SUPPORTED VERSIONS

The following table lists the supported versions of OpenShift Container Platform with Red Hat Gluster Storage Server and Container-Native Storage.

Table 5.1. Supported Versions

OpenShift Container Platform	Red Hat Gluster Storage	Container-Native Storage
3.7, 3.9	3.3.1	3.9
3.6	3.3	3.6
3.5	3.2	3.5

5.2. ENVIRONMENT REQUIREMENTS

The requirements for Red Hat Enterprise Linux Atomic Host, Red Hat OpenShift Container Platform, Red Hat Enterprise Linux, and Red Hat Gluster Storage are described in this section. A Red Hat Gluster Storage Container Native with OpenShift Container Platform environment consists of Red Hat OpenShift Container Platform installed on either Red Hat Enterprise Linux Atomic Host or Red Hat Enterprise Linux.

5.2.1. Installing Red Hat Gluster Storage Container Native with OpenShift Container Platform on Red Hat Enterprise Linux 7 based OpenShift Container Platform Cluster

This section describes the procedures to install Red Hat Gluster Storage Container Native with OpenShift Container Platform on Red Hat Enterprise Linux 7 based OpenShift Container Platform 3.9.

5.2.1.1. Setting up the Openshift Master as the Client

You can use the OpenShift Master as a client to execute the **oc** commands across the cluster when installing OpenShift. Generally, this is setup as a non-scheduled node in the cluster. This is the default configuration when using the OpenShift installer. You can also choose to install their client on their local machine to access the cluster remotely. For more information, see https://access.redhat.com/documentation/en/openshift-container-platform/3.9/single/cli-reference/#installing-the-cli.

Install heketi-client and cns-deploy packages

Execute the following commands to install **heketi-client** and the **cns-deploy** packages.

```
# subscription-manager repos --enable=rh-gluster-3-for-rhel-7-server-rpms
```

```
# yum install cns-deploy heketi-client
```

After installing the heketi-client and the cns-deploy packages, disable the gluster repo by executing the following command:

```
# subscription-manager repos --disable=rh-gluster-3-for-rhel-7-server-rpms
```

5.2.1.2. Setting up the Red Hat Enterprise Linux 7 Client for Installing Red Hat Gluster Storage Container Native with OpenShift Container Platform

To set up the Red Hat Enterprise Linux 7 client for installing Red Hat Gluster Storage Container Native with OpenShift Container Platform, perform the following steps:

Install heketi-client and cns-deploy packages

Execute the following commands to install **`heketi-client`** and the **`cns-deploy`** packages.

```
# subscription-manager repos --enable=rh-gluster-3-for-rhel-7-server-rpms
```

```
# yum install cns-deploy heketi-client
```

```
# subscription-manager repos --disable=rh-gluster-3-for-rhel-7-server-rpms
```

Subscribe to the OpenShift Container Platform 3.9 repository

If you are using OpenShift Container Platform 3.9, subscribe to 3.9 repository to enable you to install the Openshift client packages

```
# subscription-manager repos --enable=rhel-7-server-ose-3.9-rpms --
enable=rhel-7-server-rpms
```

```
# yum install atomic-openshift-clients
```

```
# yum install atomic-openshift
```

If you are installing both OpenShift Container Platform 3.9 and Container-Native Storage 3.9, proceed to Section 5.2.3, “Red Hat OpenShift Container Platform Requirements” and complete these tasks on all the OpenShift nodes (iSCSI initiator) where the app pods are hosted.

5.2.2. Installing Red Hat Gluster Storage Container Native with OpenShift Container Platform on Red Hat Enterprise Linux Atomic Host OpenShift Container Platform Cluster

Red Hat Enterprise Linux Atomic host does not support the installation of additional RPMs. Hence, an external client is required on Red Hat Enterprise Linux to install the required packages. To set up the client for Red Hat Enterprise Linux Atomic Host based installations, refer Section 5.2.1.2, “Setting up the Red Hat Enterprise Linux 7 Client for Installing Red Hat Gluster Storage Container Native with OpenShift Container Platform”

5.2.3. Red Hat OpenShift Container Platform Requirements

The following list provides the Red Hat OpenShift Container Platform requirements:

- **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 client 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. Add the following content to the **`devices`** section in the **`/etc/multipath.conf`** file

   ```
   device {
                   vendor "LIO-ORG"
                   user_friendly_names "yes" # names like mpatha
                   path_grouping_policy "failover" # one path per
   group
                   path_selector "round-robin 0"
                   failback immediate
                   path_checker "tur"
                   prio "const"
                   no_path_retry 120
                   rr_weight "uniform"
           }
   ```

4. Execute the following command to restart services:

   ```
   # systemctl restart multipathd
   ```

- The OpenShift cluster must be up and running. For information on setting up OpenShift cluster, see https://access.redhat.com/documentation/en/openshift-container-platform/3.9/paged/installation-and-configuration.

- A cluster-admin user must be created. For more information, see Appendix B, *Cluster Administrator Setup*

- All OpenShift nodes on Red Hat Enterprise Linux systems must have glusterfs-client RPMs (glusterfs, glusterfs-client-xlators, glusterfs-libs, glusterfs-fuse) installed.

- It is recommended to persist the logs for the Heketi container. For more information on persisting logs, refer https://access.redhat.com/documentation/en/openshift-container-platform/3.9/single/installation-and-configuration/#install-config-aggregate-logging.

- Next Step: If you are installing Container-Native Storage 3.9, proceed to Chapter 6, *Setting up Container-Native Storage* to complete set up tasks for port access, kernel modules, and services.

- Next Step: If you are installing Container-Ready Storage 3.9, proceed to Chapter 7, *Setting up Container-Ready Storage* to install RHGS 3.3.1 and complete set up tasks for port access, kernel modules, and services.

5.2.4. Red Hat Gluster Storage Requirements

The following list provides the details regarding the Red Hat Gluster Storage requirements:

- Installation of Heketi packages must have valid subscriptions to Red Hat Gluster Storage Server repositories.

- Red Hat Gluster Storage installations must adhere to the requirements outlined in the Red Hat Gluster Storage Installation Guide.

- The versions of Red Hat Enterprise OpenShift and Red Hat Gluster Storage integrated must be compatible, according to the information in Section 5.1, “Supported Versions” section.

- A fully-qualified domain name must be set for Red Hat Gluster Storage server node. Ensure that the correct DNS records exist, and that the fully-qualified domain name is resolvable via both forward and reverse DNS lookup.

IMPORTANT

Restrictions for using Snapshot

- After a snapshot is created, it must be accessed though 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.

5.2.5. Planning Guidelines

To prevent potential deployment or scaling issues, review the following guidelines before deploying Red Hat Container-Native Storage or Container-Ready Storage with OpenShift Container Platform..

Ensure that the Trusted Storage Pool is appropriately sized and you have room for dynamic scaling on demand. This action ensures that you do not scale beyond the following maximum limits:

- **Sizing guidelines on Container-Native Storage 3.9 or Container-Ready Storage 3.9** :

  - **Persistent volumes backed by the file interface** : For typical operations, size for 300-500 persistent volumes backed by files per three-node Container-Native Storage or Container-Ready Storage cluster. The maximum limit of supported persistent volumes backed by the file interface is 1000 persistent volumes per three-node cluster in a Container-Native Storage or Container-Ready Storage deployment. Considering that micro-services can dynamically scale as per demand, it is recommended that the initial sizing keep sufficient headroom for the scaling. If additional scaling is needed, add a new three-node Container-Native Storage or Container-Ready Storage cluster to support additional persistent volumes

    Creation of more than 1,000 persistent volumes per trusted storage pool is not supported for file-based storage.

  - **Persistent volumes backed by block-based storage** : Size for a maximum of 300 persistent volumes per three-node Container-Native Storage or Container-Ready Storage cluster. Be aware that Container-Native Storage 3.9 and Container-Ready Storage 3.9 supports only OpenShift Container Platform logging and metrics on block-backed persistent volumes.

  - **Persistent volumes backed by file and block** : Size for 300-500 persistent volumes (backed by files) and 100-200 persistent volumes (backed by block). Do not exceed these maximum limits of file or block-backed persistent volumes or the combination of a maximum 1000 persistent volumes per three-node Container-Native Storage or Container-Ready Storage cluster.
  - 3-way distributed-replicated volumes is the only supported volume type.
  - Each physical or virtual node that hosts a Red Hat Gluster Storage Container-Native Storage or Container-Ready Storage peer requires the following:
    - a minimum of 8 GB RAM and 30 MB per persistent volume.
    - the same disk type.
    - the heketidb utilises 2 GB distributed replica volume.
- **Deployment guidelines on Container-Native Storage 3.9 or Container-Ready Storage 3.9** :
  - In Container-Native Storage mode, you can install the Container-Native Storage nodes, Heketi, and all provisioner pods on OpenShift Container Platform Infrastructure nodes or OpenShift Container Platform Application nodes.
  - In Container-Ready Storage mode, you can install Heketi and all provisioners pods on OpenShift Container Platform Infrastructure nodes or on OpenShift Container Platform Application nodes
- Red Hat Gluster Storage Container Native with OpenShift Container Platform supports up to 14 snapshots per volume by default (snap-max-hard-limit =14 in Heketi Template).

CHAPTER 6. SETTING UP CONTAINER-NATIVE STORAGE

The Container-Native Storage environment addresses the use-case where applications require both shared storage and the flexibility of a converged infrastructure with compute and storage instances being scheduled and run from the same set of hardware.

6.1. CONFIGURING PORT ACCESS

- On each of the OpenShift nodes that will host the Red Hat Gluster Storage container, add the following rules to **/etc/sysconfig/iptables** in order to open the required ports:

```
-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
```

NOTE

- Port 24010 and 3260 are for gluster-blockd and iSCSI targets respectively.
- The port range starting at 49664 defines the range of ports that can be used by GlusterFS for communication to its volume bricks. In the above example the total number of bricks allowed is 512. Configure the port range based on the maximum number of bricks that could be hosted on each node.

For more information about Red Hat Gluster Storage Server ports, see https://access.redhat.com/documentation/en-us/red_hat_gluster_storage/3.3/html/administration_guide/chap-getting_started.

- Execute the following command to reload the iptables:

```
# systemctl reload iptables
```

- Execute the following command on each node to verify if the iptables are updated:

```
# iptables -L
```

6.2. ENABLING KERNEL MODULES

Before running the **cns-deploy** tool, you must ensure that the **dm_thin_pool**, **dm_multipath**, and **target_core_user** modules are loaded in the OpenShift Container Platform node. Execute the following command on all OpenShift Container Platform nodes to verify if the modules are loaded:

```
# lsmod | grep dm_thin_pool
```

```
# lsmod | grep dm_multipath
```

```
# lsmod | grep target_core_user
```

If the modules are not loaded, then execute the following command to load the modules:

```
# modprobe dm_thin_pool
```

```
# modprobe dm_multipath
```

```
# modprobe target_core_user
```

NOTE

To ensure these operations are persisted across reboots, create the following files and update each with the content as mentioned:

```
# cat /etc/modules-load.d/dm_thin_pool.conf
dm_thin_pool
```

```
# cat /etc/modules-load.d/dm_multipath.conf
dm_multipath
```

```
# cat /etc/modules-load.d/target_core_user.conf
target_core_user
```

6.3. STARTING AND ENABLING SERVICES

Execute the following commands to enable and run rpcbind on all the nodes hosting the gluster pod :

```
# systemctl add-wants multi-user rpcbind.service
# systemctl enable rpcbind.service
# systemctl start rpcbind.service
```

Execute the following command to check the status of rpcbind

```
# systemctl status rpcbind

rpcbind.service - RPC bind service
   Loaded: loaded (/usr/lib/systemd/system/rpcbind.service; enabled;
vendor preset: enabled)
   Active: active (running) since Wed 2017-08-30 21:24:21 IST; 1 day 13h
ago
 Main PID: 9945 (rpcbind)
   CGroup: /system.slice/rpcbind.service
           └─9945 /sbin/rpcbind -w
```

Next Step: Proceed to Chapter 8, *Setting up the Environment*to prepare the environment for Red Hat Gluster Storage Container Converged in OpenShift.

CHAPTER 7. SETTING UP CONTAINER-READY STORAGE

In a Container-Ready Storage set-up a dedicated Red Hat Gluster Storage cluster is available external to the OpenShift Container Platform. The storage is provisioned from the Red Hat Gluster Storage cluster.

7.1. INSTALLING RED HAT GLUSTER STORAGE SERVER ON RED HAT ENTERPRISE LINUX (LAYERED INSTALL)

Layered install involves installing Red Hat Gluster Storage over Red Hat Enterprise Linux.

IMPORTANT

It is recommended to create a separate `/var` partition that is large enough (50GB - 100GB) for log files, geo-replication related miscellaneous files, and other files.

1. **Perform a base install of Red Hat Enterprise Linux 7 Server**
   Container-Ready Storage is supported only on Red Hat Enterprise Linux 7.

2. **Register the System with Subscription Manager**
   Run the following command and enter your Red Hat Network user name and password to register the system with the Red Hat Network:

   ```
   # subscription-manager register
   ```

3. **Identify Available Entitlement Pools**
   Run the following commands to find entitlement pools containing the repositories required to install Red Hat Gluster Storage:

   ```
   # subscription-manager list --available
   ```

4. **Attach Entitlement Pools to the System**
   Use the pool identifiers located in the previous step to attach the **`Red Hat Enterprise Linux Server`** and **`Red Hat Gluster Storage`** entitlements to the system. Run the following command to attach the entitlements:

   ```
   # subscription-manager attach --pool=[POOLID]
   ```

   For example:

   ```
   # subscription-manager attach --
   pool=8a85f9814999f69101499c05aa706e47
   ```

5. **Enable the Required Channels**

   For Red Hat Gluster Storage 3.3 on Red Hat Enterprise Linux 7.x

   1. Run the following commands to enable the repositories required to install Red Hat Gluster Storage

```
# subscription-manager repos --enable=rhel-7-server-rpms
# subscription-manager repos --enable=rh-gluster-3-for-rhel-7-
server-rpms
```

6. **Verify if the Channels are Enabled**
   Run the following command to verify if the channels are enabled:

   ```
   # yum repolist
   ```

7. **Update all packages**
   Ensure that all packages are up to date by running the following command.

   ```
   # yum update
   ```

   IMPORTANT

   If any kernel packages are updated, reboot the system with the following command.

   ```
   # shutdown -r now
   ```

8. **Kernel Version Requirement**
   Container-Ready Storage requires the kernel-3.10.0-690.el7 version or higher to be used on the system. Verify the installed and running kernel versions by running the following command:

   ```
   # rpm -q kernel
   kernel-3.10.0-693.el7.x86_64
   ```

   ```
   # uname -r
   3.10.0-693.el7.x86_64
   ```

9. **Install Red Hat Gluster Storage**
   Run the following command to install Red Hat Gluster Storage:

   ```
   # yum install redhat-storage-server
   ```

   1. To enable gluster-block execute the following command:

      ```
      # yum install gluster-block
      ```

10. **Reboot**
    Reboot the system.

7.2. CONFIGURING PORT ACCESS

This section provides information about the ports that must be open for Container-Ready Storage .

Red Hat Gluster Storage Server uses the listed ports. You must ensure that the firewall settings do not prevent access to these ports.

Execute the following commands to open the required ports for both runtime and permanent configurations on all Red Hat Gluster Storage nodes:

```
# firewall-cmd --zone=zone_name --add-port=24010/tcp --add-port=3260/tcp -
-add-port=111/tcp --add-port=22/tcp --add-port=24007/tcp --add-
port=24008/tcp --add-port=49152-49664/tcp
# firewall-cmd --zone=zone_name --add-port=24010/tcp --add-port=3260/tcp -
-add-port=111/tcp --add-port=22/tcp --add-port=24007/tcp --add-
port=24008/tcp --add-port=49152-49664/tcp --permanent
```

NOTE

- Port 24010 and 3260 are for gluster-blockd and iSCSI targets respectively.
- The port range starting at 49664 defines the range of ports that can be used by GlusterFS for communication to its volume bricks. In the above example the total number of bricks allowed is 512. Configure the port range based on the maximum number of bricks that could be hosted on each node.

7.3. ENABLING KERNEL MODULES

Execute the following commands to enable kernel modules:

1. You must ensure that the **`dm_thin_pool`** and **`target_core_user`** modules are loaded in the Red Hat Gluster Storage nodes.

   ```
   # modprobe target_core_user
   ```

   ```
   # modprobe dm_thin_pool
   ```

   Execute the following command to verify if the modules are loaded:

   ```
   # lsmod | grep dm_thin_pool
   ```

   ```
   # lsmod | grep target_core_user
   ```

   NOTE

   To ensure these operations are persisted across reboots, create the following files and update each file with the content as mentioned:

   ```
   # cat /etc/modules-load.d/dm_thin_pool.conf
   dm_thin_pool
   ```

   ```
   # cat /etc/modules-load.d/target_core_user.conf
   target_core_user
   ```

2. You must ensure that the **`dm_multipath`** module is loaded on all OpenShift Container Platform nodes.

```
# modprobe dm_multipath
```

Execute the following command to verify if the modules are loaded:

```
# lsmod | grep dm_multipath
```

NOTE

To ensure these operations are persisted across reboots, create the following file and update it with the content as mentioned:

```
# cat /etc/modules-load.d/dm_multipath.conf
dm_multipath
```

7.4. STARTING AND ENABLING SERVICES

Execute the following commands to start glusterd and gluster-blockd:

```
# systemctl start sshd
```

```
# systemctl enable sshd
```

```
# systemctl start glusterd
```

```
# systemctl enable glusterd
```

```
# systemctl start gluster-blockd
```

```
# systemctl enable gluster-blockd
```

Next Step: Proceed to Chapter 8, *Setting up the Environment*to prepare the environment for Red Hat Gluster Storage Container Converged in OpenShift.

CHAPTER 8. SETTING UP THE ENVIRONMENT

This chapter outlines the details for setting up the environment for Red Hat Gluster Storage Container Converged in OpenShift.

8.1. PREPARING THE RED HAT OPENSHIFT CONTAINER PLATFORM CLUSTER

Execute the following steps to prepare the Red Hat OpenShift Container Platform cluster:

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://dhcp46-
   24.lab.eng.blr.redhat.com:8443 (openshift)
   Username: test
   Password:
   Login successful.

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

     * default
       kube-system
       logging
       management-infra
       openshift
       openshift-infra


   Using project "default".
   ```

2. On the master or client, execute the following command to create a project, which will contain all the containerized Red Hat Gluster Storage services:

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

   For example:

   ```
   # oc new-project storage-project

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

3. 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 default
   ```

4. Execute the following steps on the master to set up the router:

   NOTE

   If a router already exists, proceed to Step 5. To verify if the router is already deployed, execute the following command:

   ```
   # oc get dc --all-namespaces
   ```

   1. Execute the following command to enable the deployment of the router:

      ```
      # oc adm policy add-scc-to-user privileged -z router
      ```

   2. Execute the following command to deploy the router:

      ```
      # oc adm router storage-project-router --replicas=1
      ```

   3. Edit the subdomain name in the config.yaml file located at **`/etc/origin/master/master-config.yaml`**.

      For example:

      ```
      subdomain: "cloudapps.mystorage.com"
      ```

      For more information, refer https://access.redhat.com/documentation/en-us/openshift_container_platform/3.9/html-single/installation_and_configuration/#customizing-the-default-routing-subdomain.

   4. For OpenShift Container Platform 3.7 and 3.9 execute the following command to restart the services :

      ```
      # systemctl restart atomic-openshift-master-api atomic-openshift-
      master-controllers
      ```

      NOTE

      If the router setup fails, use the port forward method as described in Appendix C, *Client Configuration using Port Forwarding*.

   For more information regarding router setup, see https://access.redhat.com/documentation/en/openshift-container-platform/3.9/paged/installation-and-configuration/chapter-4-setting-up-a-router

5. Execute the following command to verify if the router is running:

   ```
   # oc get dc <router_name>
   ```

   For example:

```
# oc get dc storage-project-router
NAME                      REVISION   DESIRED    CURRENT    TRIGGERED BY
storage-project-router    1           1          1          config
```

NOTE

Ensure you do not edit the **`/etc/dnsmasq.conf`** file until the router has started.

6. After the router is running, the client has to be setup to access the services in the OpenShift cluster. Execute the following steps on the client to set up the DNS.

   1. Execute the following command to find the IP address of the router:

      ```
      # oc get pods -o wide --all-namespaces | grep router
      storage-project storage-project-router-1-cm874        1/1
      Running   119d       10.70.43.132   dhcp43-
      132.lab.eng.blr.redhat.com
      ```

   2. Edit the /etc/dnsmasq.conf file and add the following line to the file:

      ```
      address=/.cloudapps.mystorage.com/<Router_IP_Address>
      ```

      where, *Router_IP_Address* is the IP address of the node where the router is running.

   3. Restart the **dnsmasq** service by executing the following command:

      ```
      # systemctl restart dnsmasq
      ```

   4. Edit /etc/resolv.conf and add the following line:

      ```
      nameserver 127.0.0.1
      ```

   For more information regarding setting up the DNS, see https://access.redhat.com/documentation/en/openshift-container-platform/3.9/single/installation-and-configuration/#envirornment-requirements.

Next step: If you are installing Container-Native Storage 3.9, proceed to the next section, Section 8.2, “ Deploying Containerized Red Hat Gluster Storage Solutions”.

8.2. DEPLOYING CONTAINERIZED RED HAT GLUSTER STORAGE SOLUTIONS

The following section covers deployment of the Container-Native Storage pods and Container-Ready Storage and using the **`cns-deploy`** tool.

NOTE

- It is recommended that a separate cluster for OpenShift Container Platform infrastructure workload (registry, logging and metrics) and application pod storage. Hence, if you have more than 6 nodes ensure you create multiple clusters with a minimum of 3 nodes each. The infrastructure cluster should belong to the `default` project namespace.

- If you want to enable encryption on the Container-Native Storage setup, refer Chapter 17, *Enabling Encryption* before proceeding with the following steps.

1. You must first provide a topology file for heketi which describes the topology of the Red Hat Gluster Storage nodes and their attached storage devices. A sample, formatted topology file (topology-sample.json) is installed with the ‘heketi-client’ package in the /usr/share/heketi/ directory.

```
{
    "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

NOTE

Copy the topology file from the default location to your location and then edit it:

```
# cp /usr/share/heketi/topology-sample.json
/<Path>/topology.json
```

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.

IMPORTANT

Heketi stores its database on a Red Hat Gluster Storage volume. In cases where the volume is down, the Heketi service does not respond due to the unavailability of the volume served by a disabled trusted storage pool. To resolve this issue, restart the trusted storage pool which contains the Heketi volume.

To deploy Container-Native Storage, refer Section 8.2.1, “Deploying Container-Native Storage”. To deploy Container-Ready Storage refer Section 8.2.2, “Deploying Container-Ready Storage”.

8.2.1. Deploying Container-Native Storage

Execute the following commands to deploy container-native storage:

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

```
# cns-deploy -n <namespace> -g --admin-key <Key> topology.json
```

NOTE

- From Container-Native Storage 3.6, support for S3 compatible Object Store in Container-Native Storage is under technology preview. To deploy S3 compatible object store in Container-Native Storage see Step 1a below.
- In the above command, the value for **admin-key** is the secret string for heketi admin user. The heketi administrator will have access to all APIs and commands. Default is to use no secret.
- The **BLOCK_HOST_SIZE** parameter in cns-deploy controls the size (in GB) of the automatically created Red Hat Gluster Storage volumes hosting the gluster-block volumes (For more information, see Section 9.2, “Block Storage”). This default configuration will dynamically create block-hosting volumes of 500GB in size when more space is required. If you want to change this value then use --block-host in cns-deploy. For example:

```
# cns-deploy -n storage-project -g --admin-key secret
--block-host 1000 topology.json
```

For example:

```
# cns-deploy -n storage-project -g --admin-key secret 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'

Each of the nodes that will host GlusterFS must also have
appropriate firewall
rules for the required GlusterFS ports:
 * 111   - rpcbind (for glusterblock)
 * 2222  - sshd (if running GlusterFS in a pod)
 * 3260  - iSCSI targets (for glusterblock)
 * 24006 - glusterblockd
 * 24007 - GlusterFS Management
 * 24008 - GlusterFS RDMA
 * 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.

The following kernel modules must be loaded:
 * dm_snapshot
 * dm_mirror
 * dm_thin_pool
 * dm_multipath
 * target_core_user

For systems with SELinux, the following settings need to be
considered:
 * virt_sandbox_use_fusefs should be enabled on each node to allow
writing to
   remote GlusterFS volumes

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.
Using namespace "storage-project".
Checking for pre-existing resources...
  GlusterFS pods ... not found.
  deploy-heketi pod ... not found.
  heketi pod ... not found.
  glusterblock-provisioner pod ... not found.
  gluster-s3 pod ... not found.
```

```
Creating initial resources ... template "deploy-heketi" created
serviceaccount "heketi-service-account" created
template "heketi" created
template "glusterfs" created
role "edit" added: "system:serviceaccount:storage-project:heketi-
service-account"
OK
node "ip-172-18-5-29.ec2.internal" labeled
node "ip-172-18-8-205.ec2.internal" labeled
node "ip-172-18-6-100.ec2.internal" labeled
daemonset "glusterfs" created
Waiting for GlusterFS pods to start ... OK
secret "heketi-config-secret" created
secret "heketi-config-secret" labeled
service "deploy-heketi" created
route "deploy-heketi" created
deploymentconfig "deploy-heketi" created
Waiting for deploy-heketi pod to start ... OK
Creating cluster ... ID: 30cd12e60f860fce21e7e7457d07db36
Allowing file volumes on cluster.
Allowing block volumes on cluster.
Creating node ip-172-18-5-29.ec2.internal ... ID:
4077242c76e5f477a27c5c47247cb348
Adding device /dev/xvdc ... OK
Creating node ip-172-18-8-205.ec2.internal ... ID:
dda0e7d568d7b2f76a7e7491cfc26dd3
Adding device /dev/xvdc ... OK
Creating node ip-172-18-6-100.ec2.internal ... ID:
30a1795ca515c85dca32b09be7a68733
Adding device /dev/xvdc ... OK
heketi topology loaded.
Saving /tmp/heketi-storage.json
secret "heketi-storage-secret" created
endpoints "heketi-storage-endpoints" created
service "heketi-storage-endpoints" created
job "heketi-storage-copy-job" created
service "heketi-storage-endpoints" labeled
deploymentconfig "deploy-heketi" deleted
route "deploy-heketi" deleted
service "deploy-heketi" deleted
job "heketi-storage-copy-job" deleted
pod "deploy-heketi-1-frjpt" deleted
secret "heketi-storage-secret" deleted
template "deploy-heketi" deleted
service "heketi" created
route "heketi" created
deploymentconfig "heketi" created
Waiting for heketi pod to start ... OK

heketi is now running and accessible via http://heketi-storage-
project.cloudapps.mystorage.com . To run
administrative commands you can install 'heketi-cli' and use it as
follows:

  # heketi-cli -s http://heketi-storage-
project.cloudapps.mystorage.com --user admin --secret '<ADMIN_KEY>'
```

```
cluster list

You can find it at https://github.com/heketi/heketi/releases .
Alternatively,
use it from within the heketi pod:

  # /bin/oc -n storage-project exec -it <HEKETI_POD> -- heketi-cli -
s http://localhost:8080 --user admin --secret '<ADMIN_KEY>' cluster
list

For dynamic provisioning, create a StorageClass similar to this:

---
apiVersion: storage.k8s.io/v1beta1
kind: StorageClass
metadata:
  name: glusterfs-storage
provisioner: kubernetes.io/glusterfs
parameters:
  resturl: "http://heketi-storage-project.cloudapps.mystorage.com"

Ready to create and provide GlusterFS volumes.
clusterrole "glusterblock-provisioner-runner" created
serviceaccount "glusterblock-provisioner" created
clusterrolebinding "glusterblock-provisioner" created
deploymentconfig "glusterblock-provisioner-dc" created
Waiting for glusterblock-provisioner pod to start ... OK
Ready to create and provide Gluster block volumes.

Deployment complete!
```

NOTE

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

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

1. To deploy S3 compatible object store along with Heketi and Red Hat Gluster Storage pods, execute the following command:

```
#  cns-deploy /opt/topology.json --deploy-gluster  --namespace
<namespace> --yes --admin-key <key>  --log-file=<path/to/logfile>
--object-account <object account name> --object-user <object user
name>  --object-password <object user password> --verbose
```

`object-account`, **`object-user`**, and **`object-password`** are required credentials for deploying the gluster-s3 container. If any of these are missing, gluster-s3 container deployment will be skipped.

`object-sc` and **`object-capacity`** are optional parameters. Where, **`object-sc`** is used to specify a pre-existing StorageClass to use to create Red Hat Gluster Storage volumes to back the object store and **`object-capacity`** is the total capacity of the Red Hat Gluster Storage volume which will store the object data.

For example:

```
#  cns-deploy /opt/topology.json --deploy-gluster  --namespace
storage-project --yes --admin-key secret --log-file=/var/log/cns-
deploy/444-cns-deploy.log --object-account testvolume --object-
user adminuser --object-password itsmine --verbose
Using OpenShift CLI.

Checking status of namespace matching 'storage-project':
storage-project   Active    56m
Using namespace "storage-project".
Checking for pre-existing resources...
  GlusterFS pods ...
Checking status of pods matching '--selector=glusterfs=pod':
No resources found.
Timed out waiting for pods matching '--selector=glusterfs=pod'.
not found.
  deploy-heketi pod ...
Checking status of pods matching '--selector=deploy-heketi=pod':
No resources found.
Timed out waiting for pods matching '--selector=deploy-
heketi=pod'.
not found.
  heketi pod ...
Checking status of pods matching '--selector=heketi=pod':
No resources found.
Timed out waiting for pods matching '--selector=heketi=pod'.
not found.
  glusterblock-provisioner pod ...
Checking status of pods matching '--selector=glusterfs=block-
provisioner-pod':
No resources found.
Timed out waiting for pods matching '--selector=glusterfs=block-
provisioner-pod'.
not found.
  gluster-s3 pod ...
Checking status of pods matching '--selector=glusterfs=s3-pod':
No resources found.
Timed out waiting for pods matching '--selector=glusterfs=s3-
pod'.
not found.
Creating initial resources ... /usr/bin/oc -n storage-project
create -f /usr/share/heketi/templates/deploy-heketi-template.yaml
2>&1
template "deploy-heketi" created
/usr/bin/oc -n storage-project create -f
/usr/share/heketi/templates/heketi-service-account.yaml 2>&1
serviceaccount "heketi-service-account" created
/usr/bin/oc -n storage-project create -f
/usr/share/heketi/templates/heketi-template.yaml 2>&1
template "heketi" created
/usr/bin/oc -n storage-project create -f
/usr/share/heketi/templates/glusterfs-template.yaml 2>&1
template "glusterfs" created
/usr/bin/oc -n storage-project policy add-role-to-user edit
system:serviceaccount:storage-project:heketi-service-account 2>&1
```

```
role "edit" added: "system:serviceaccount:storage-project:heketi-
service-account"
/usr/bin/oc -n storage-project adm policy add-scc-to-user
privileged -z heketi-service-account
OK
Marking 'dhcp46-122.lab.eng.blr.redhat.com' as a GlusterFS node.
/usr/bin/oc -n storage-project label nodes dhcp46-
122.lab.eng.blr.redhat.com storagenode=glusterfs 2>&1
node "dhcp46-122.lab.eng.blr.redhat.com" labeled
Marking 'dhcp46-9.lab.eng.blr.redhat.com' as a GlusterFS node.
/usr/bin/oc -n storage-project label nodes dhcp46-
9.lab.eng.blr.redhat.com storagenode=glusterfs 2>&1
node "dhcp46-9.lab.eng.blr.redhat.com" labeled
Marking 'dhcp46-134.lab.eng.blr.redhat.com' as a GlusterFS node.
/usr/bin/oc -n storage-project label nodes dhcp46-
134.lab.eng.blr.redhat.com storagenode=glusterfs 2>&1
node "dhcp46-134.lab.eng.blr.redhat.com" labeled
Deploying GlusterFS pods.
/usr/bin/oc -n storage-project process -p NODE_LABEL=glusterfs
glusterfs | /usr/bin/oc -n storage-project create -f - 2>&1
daemonset "glusterfs" created
Waiting for GlusterFS pods to start ...
Checking status of pods matching '--selector=glusterfs=pod':
glusterfs-6fj2v   1/1       Running   0          52s
glusterfs-ck40f   1/1       Running   0          52s
glusterfs-kbtz4   1/1       Running   0          52s
OK
/usr/bin/oc -n storage-project create secret generic heketi-
config-secret --from-file=private_key=/dev/null --from-
file=./heketi.json --from-file=topology.json=/opt/topology.json
secret "heketi-config-secret" created
/usr/bin/oc -n storage-project label --overwrite secret heketi-
config-secret glusterfs=heketi-config-secret heketi=config-secret
secret "heketi-config-secret" labeled
/usr/bin/oc -n storage-project process -p
HEKETI_EXECUTOR=kubernetes -p HEKETI_FSTAB=/var/lib/heketi/fstab
-p HEKETI_ADMIN_KEY= -p HEKETI_USER_KEY= deploy-heketi |
/usr/bin/oc -n storage-project create -f - 2>&1
service "deploy-heketi" created
route "deploy-heketi" created
deploymentconfig "deploy-heketi" created
Waiting for deploy-heketi pod to start ...
Checking status of pods matching '--selector=deploy-heketi=pod':
deploy-heketi-1-hf9rn   1/1       Running   0          2m
OK
Determining heketi service URL ... OK
/usr/bin/oc -n storage-project exec -it deploy-heketi-1-hf9rn --
heketi-cli -s http://localhost:8080 --user admin --secret ''
topology load --json=/etc/heketi/topology.json 2>&1
Creating cluster ... ID: 252509038eb8568162ec5920c12bc243
Allowing file volumes on cluster.
Allowing block volumes on cluster.
Creating node dhcp46-122.lab.eng.blr.redhat.com ... ID:
73ad287ae1ef231f8a0db46422367c9a
Adding device /dev/sdd ... OK
Adding device /dev/sde ... OK
Adding device /dev/sdf ... OK
Creating node dhcp46-9.lab.eng.blr.redhat.com ... ID:
0da1b20daaad2d5c57dbfc4f6ab78001
Adding device /dev/sdd ... OK
Adding device /dev/sde ... OK
Adding device /dev/sdf ... OK
Creating node dhcp46-134.lab.eng.blr.redhat.com ... ID:
4b3b62fc0efd298dedbcdacf0b498e65
Adding device /dev/sdd ... OK
Adding device /dev/sde ... OK
Adding device /dev/sdf ... OK
heketi topology loaded.
/usr/bin/oc -n storage-project exec -it deploy-heketi-1-hf9rn --
heketi-cli -s http://localhost:8080 --user admin --secret ''
setup-openshift-heketi-storage --listfile=/tmp/heketi-
storage.json --image rhgs3/rhgs-volmanager-rhel7:3.3.0-17 2>&1
Saving /tmp/heketi-storage.json
/usr/bin/oc -n storage-project exec -it deploy-heketi-1-hf9rn --
cat /tmp/heketi-storage.json | /usr/bin/oc -n storage-project
create -f - 2>&1
secret "heketi-storage-secret" created
endpoints "heketi-storage-endpoints" created
service "heketi-storage-endpoints" created
job "heketi-storage-copy-job" created

Checking status of pods matching '--selector=job-name=heketi-
storage-copy-job':
heketi-storage-copy-job-87v6n   0/1       Completed   0
7s
/usr/bin/oc -n storage-project label --overwrite svc heketi-
storage-endpoints glusterfs=heketi-storage-endpoints
heketi=storage-endpoints
service "heketi-storage-endpoints" labeled
/usr/bin/oc -n storage-project delete
all,service,jobs,deployment,secret --selector="deploy-heketi"
2>&1
deploymentconfig "deploy-heketi" deleted
route "deploy-heketi" deleted
service "deploy-heketi" deleted
job "heketi-storage-copy-job" deleted
pod "deploy-heketi-1-hf9rn" deleted
secret "heketi-storage-secret" deleted
/usr/bin/oc -n storage-project delete dc,route,template --
selector="deploy-heketi" 2>&1
template "deploy-heketi" deleted
/usr/bin/oc -n storage-project process -p
HEKETI_EXECUTOR=kubernetes -p HEKETI_FSTAB=/var/lib/heketi/fstab
-p HEKETI_ADMIN_KEY= -p HEKETI_USER_KEY= heketi | /usr/bin/oc -n
storage-project create -f - 2>&1
service "heketi" created
route "heketi" created
deploymentconfig "heketi" created
Waiting for heketi pod to start ...
Checking status of pods matching '--selector=heketi=pod':
heketi-1-zzblp   1/1       Running   0         31s
OK
```

```
Determining heketi service URL ... OK

heketi is now running and accessible via http://heketi-storage-
project.cloudapps.mystorage.com . To run
administrative commands you can install 'heketi-cli' and use it as
follows:

  # heketi-cli -s http://heketi-storage-
project.cloudapps.mystorage.com --user admin --secret
'<ADMIN_KEY>' cluster list

You can find it at https://github.com/heketi/heketi/releases .
Alternatively,
use it from within the heketi pod:

  # /usr/bin/oc -n storage-project exec -it <HEKETI_POD> --
heketi-cli -s http://localhost:8080 --user admin --secret
'<ADMIN_KEY>' cluster list

For dynamic provisioning, create a StorageClass similar to this:

---
apiVersion: storage.k8s.io/v1beta1
kind: StorageClass
metadata:
  name: glusterfs-storage
provisioner: kubernetes.io/glusterfs
parameters:
  resturl: "http://heketi-storage-
project.cloudapps.mystorage.com"

Ready to create and provide GlusterFS volumes.
sed -e 's/\${NAMESPACE}/storage-project/'
/usr/share/heketi/templates/glusterblock-provisioner.yaml |
/usr/bin/oc -n storage-project create -f - 2>&1
clusterrole "glusterblock-provisioner-runner" created
serviceaccount "glusterblock-provisioner" created
clusterrolebinding "glusterblock-provisioner" created
deploymentconfig "glusterblock-provisioner-dc" created
Waiting for glusterblock-provisioner pod to start ...
Checking status of pods matching '--selector=glusterfs=block-
provisioner-pod':
glusterblock-provisioner-dc-1-xm6bv   1/1       Running   0
6s
OK
Ready to create and provide Gluster block volumes.
/usr/bin/oc -n storage-project create secret generic heketi-
storage-project-admin-secret --from-literal=key= --
type=kubernetes.io/glusterfs
secret "heketi-storage-project-admin-secret" created
/usr/bin/oc -n storage-project label --overwrite secret heketi-
storage-project-admin-secret glusterfs=s3-heketi-storage-project-
admin-secret gluster-s3=heketi-storage-project-admin-secret
secret "heketi-storage-project-admin-secret" labeled
sed -e 's/\${STORAGE_CLASS}/glusterfs-for-s3/' -e
's/\${HEKETI_URL}/heketi-storage-
```

```
project.cloudapps.mystorage.com/' -e 's/\${NAMESPACE}/storage-
project/' /usr/share/heketi/templates/gluster-s3-
storageclass.yaml | /usr/bin/oc -n storage-project create -f -
2>&1
storageclass "glusterfs-for-s3" created
sed -e 's/\${STORAGE_CLASS}/glusterfs-for-s3/' -e
's/\${VOLUME_CAPACITY}/2Gi/' /usr/share/heketi/templates/gluster-
s3-pvcs.yaml | /usr/bin/oc -n storage-project create -f - 2>&1
persistentvolumeclaim "gluster-s3-claim" created
persistentvolumeclaim "gluster-s3-meta-claim" created

Checking status of persistentvolumeclaims matching '--
selector=glusterfs in (s3-pvc, s3-meta-pvc)':
gluster-s3-claim        Bound     pvc-35b6c1f0-9c65-11e7-9c8c-
005056b3ded1   2Gi       RWX       glusterfs-for-s3   18s
gluster-s3-meta-claim   Bound     pvc-35b86e7a-9c65-11e7-9c8c-
005056b3ded1   1Gi       RWX       glusterfs-for-s3   18s
/usr/bin/oc -n storage-project create -f
/usr/share/heketi/templates/gluster-s3-template.yaml 2>&1
template "gluster-s3" created
/usr/bin/oc -n storage-project process -p S3_ACCOUNT=testvolume -
p S3_USER=adminuser -p S3_PASSWORD=itsmine gluster-s3 |
/usr/bin/oc -n storage-project create -f - 2>&1
service "gluster-s3-service" created
route "gluster-s3-route" created
deploymentconfig "gluster-s3-dc" created
Waiting for gluster-s3 pod to start ...
Checking status of pods matching '--selector=glusterfs=s3-pod':
gluster-s3-dc-1-x3x4q   1/1       Running   0          6s
OK
Ready to create and provide Gluster object volumes.

Deployment complete!
```

2. Execute the following command to let the client communicate with the container:

   ```
   # export  HEKETI_CLI_SERVER=http://heketi-<project_name>.
   <sub_domain_name>
   ```

   For example:

   ```
   # export  HEKETI_CLI_SERVER=http://heketi-storage-
   project.cloudapps.mystorage.com
   ```

   To verify if Heketi is loaded with the topology execute the following command:

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

NOTE

The cns-deploy tool does not support scaling up of the cluster. To manually scale-up the cluster, refer Chapter 12, *Managing Clusters*

Next step: If you are installing Container-Ready Storage 3.9, proceed to Chapter 10, *Updating the*

Registry with Container-Native Storage as the Storage Back-end.

8.2.2. Deploying Container-Ready Storage

Execute the following commands to deploy container-ready storage:

1. To set a passwordless SSH to all Red Hat Gluster Storage nodes, execute the following command on the client for each of the Red Hat Gluster Storage node:

   ```
   # ssh-copy-id -i /root/.ssh/id_rsa root@<ip/hostname_rhgs node>
   ```

2. Execute the following command on the client to deploy heketi pod and to create a cluster of Red Hat Gluster Storage nodes:

   ```
   # cns-deploy -n <namespace> --admin-key <Key>  -s /root/.ssh/id_rsa
   topology.json
   ```

   NOTE

   - Support for S3 compatible Object Store is under technology preview. To deploy S3 compatible object store see Step 2a below.
   - In the above command, the value for **admin-key** is the secret string for heketi admin user. The heketi administrator will have access to all APIs and commands. Default is to use no secret.
   - The **BLOCK_HOST_SIZE** parameter in cns-deploy controls the size (in GB) of the automatically created Red Hat Gluster Storage volumes hosting the gluster-block volumes (For more information, see Section 9.2, “Block Storage”). This default configuration will dynamically create block-hosting volumes of 500GB in size when more space is required. If you want to change this value then use --block-host in cns-deploy. For example:

     ```
     # cns-deploy -n storage-project -g --admin-key secret
     --block-host 1000 topology.json
     ```

   For example:

   ```
   # cns-deploy -n storage-project --admin-key secret -s
   /root/.ssh/id_rsa 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'

   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 Management
 * 24008 - GlusterFS RDMA
 * 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.

The following kernel modules must be loaded:
 * dm_snapshot
 * dm_mirror
 * dm_thin_pool

For systems with SELinux, the following settings need to be
considered:
 * virt_sandbox_use_fusefs should be enabled on each node to allow
writing to
   remote GlusterFS volumes

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.
Using namespace "storage-project".
Checking for pre-existing resources...
  GlusterFS pods ... not found.
  deploy-heketi pod ... not found.
  heketi pod ... not found.
Creating initial resources ... template "deploy-heketi" created
serviceaccount "heketi-service-account" created
template "heketi" created
role "edit" added: "system:serviceaccount:storage-project:heketi-
service-account"
OK
secret "heketi-config-secret" created
secret "heketi-config-secret" labeled
service "deploy-heketi" created
route "deploy-heketi" created
deploymentconfig "deploy-heketi" created
Waiting for deploy-heketi pod to start ... OK
Creating cluster ... ID: 60bf06636eb4eb81d4e9be4b04cfce92
Allowing file volumes on cluster.
Allowing block volumes on cluster.
Creating node dhcp47-104.lab.eng.blr.redhat.com ... ID:
eadc66f9d03563bcfc3db3fe636c34be
Adding device /dev/sdd ... OK
Adding device /dev/sde ... OK
Adding device /dev/sdf ... OK
Creating node dhcp47-83.lab.eng.blr.redhat.com ... ID:
178684b0a0425f51b8f1a032982ffe4d
Adding device /dev/sdd ... OK
Adding device /dev/sde ... OK
Adding device /dev/sdf ... OK
Creating node dhcp46-152.lab.eng.blr.redhat.com ... ID:
08cd7034ef7ac66499dc040d93cf4a93
Adding device /dev/sdd ... OK
Adding device /dev/sde ... OK
Adding device /dev/sdf ... OK
heketi topology loaded.
Saving /tmp/heketi-storage.json
secret "heketi-storage-secret" created
endpoints "heketi-storage-endpoints" created
service "heketi-storage-endpoints" created
job "heketi-storage-copy-job" created
service "heketi-storage-endpoints" labeled
deploymentconfig "deploy-heketi" deleted
route "deploy-heketi" deleted
service "deploy-heketi" deleted
job "heketi-storage-copy-job" deleted
pod "deploy-heketi-1-30c06" deleted
secret "heketi-storage-secret" deleted
template "deploy-heketi" deleted
service "heketi" created
route "heketi" created
deploymentconfig "heketi" created
Waiting for heketi pod to start ... OK

heketi is now running and accessible via http://heketi-storage-
project.cloudapps.mystorage.com . To run
administrative commands you can install 'heketi-cli' and use it as
follows:

  # heketi-cli -s http://heketi-storage-
project.cloudapps.mystorage.com --user admin --secret '<ADMIN_KEY>'
cluster list

You can find it at https://github.com/heketi/heketi/releases .
Alternatively,
use it from within the heketi pod:

  # /usr/bin/oc -n storage-project exec -it <HEKETI_POD> -- heketi-
cli -s http://localhost:8080 --user admin --secret '<ADMIN_KEY>'
cluster list

For dynamic provisioning, create a StorageClass similar to this:

---
apiVersion: storage.k8s.io/v1beta1
```

```
kind: StorageClass
metadata:
  name: glusterfs-storage
provisioner: kubernetes.io/glusterfs
parameters:
  resturl: "http://heketi-storage-project.cloudapps.mystorage.com"


Deployment complete!
```

NOTE

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

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

1. To deploy S3 compatible object store along with Heketi and Red Hat Gluster Storage pods, execute the following command:

   ```
   #  cns-deploy /opt/topology.json --deploy-gluster  --namespace
   <namespace> --admin-key <Key> --yes --log-file=<path/to/logfile>
   --object-account <object account name> --object-user <object user
   name>  --object-password <object user password> --verbose
   ```

   `object-account`, **`object-user`**, and **`object-password`** are required credentials for deploying the gluster-s3 container. If any of these are missing, gluster-s3 container deployment will be skipped.

   `object-sc` and **`object-capacity`** are optional parameters. Where, **`object-sc`** is used to specify a pre-existing StorageClass to use to create Red Hat Gluster Storage volumes to back the object store and **`object-capacity`** is the total capacity of the Red Hat Gluster Storage volume which will store the object data.

   For example:

   ```
   #  cns-deploy /opt/topology.json --deploy-gluster  --namespace
   storage-project --admin-key secret --yes --log-file=/var/log/cns-
   deploy/444-cns-deploy.log --object-account testvolume --object-
   user adminuser --object-password itsmine --verbose
   Using OpenShift CLI.

   Checking status of namespace matching 'storage-project':
   storage-project   Active    56m
   Using namespace "storage-project".
   Checking for pre-existing resources...
     GlusterFS pods ...
   Checking status of pods matching '--selector=glusterfs=pod':
   No resources found.
   Timed out waiting for pods matching '--selector=glusterfs=pod'.
   not found.
     deploy-heketi pod ...
   Checking status of pods matching '--selector=deploy-heketi=pod':
   No resources found.
   ```

```
Timed out waiting for pods matching '--selector=deploy-
heketi=pod'.
not found.
  heketi pod ...
Checking status of pods matching '--selector=heketi=pod':
No resources found.
Timed out waiting for pods matching '--selector=heketi=pod'.
not found.
  glusterblock-provisioner pod ...
Checking status of pods matching '--selector=glusterfs=block-
provisioner-pod':
No resources found.
Timed out waiting for pods matching '--selector=glusterfs=block-
provisioner-pod'.
not found.
  gluster-s3 pod ...
Checking status of pods matching '--selector=glusterfs=s3-pod':
No resources found.
Timed out waiting for pods matching '--selector=glusterfs=s3-
pod'.
not found.
Creating initial resources ... /usr/bin/oc -n storage-project
create -f /usr/share/heketi/templates/deploy-heketi-template.yaml
2>&1
template "deploy-heketi" created
/usr/bin/oc -n storage-project create -f
/usr/share/heketi/templates/heketi-service-account.yaml 2>&1
serviceaccount "heketi-service-account" created
/usr/bin/oc -n storage-project create -f
/usr/share/heketi/templates/heketi-template.yaml 2>&1
template "heketi" created
/usr/bin/oc -n storage-project create -f
/usr/share/heketi/templates/glusterfs-template.yaml 2>&1
template "glusterfs" created
/usr/bin/oc -n storage-project policy add-role-to-user edit
system:serviceaccount:storage-project:heketi-service-account 2>&1
role "edit" added: "system:serviceaccount:storage-project:heketi-
service-account"
/usr/bin/oc -n storage-project adm policy add-scc-to-user
privileged -z heketi-service-account
OK
Marking 'dhcp46-122.lab.eng.blr.redhat.com' as a GlusterFS node.
/usr/bin/oc -n storage-project label nodes dhcp46-
122.lab.eng.blr.redhat.com storagenode=glusterfs 2>&1
node "dhcp46-122.lab.eng.blr.redhat.com" labeled
Marking 'dhcp46-9.lab.eng.blr.redhat.com' as a GlusterFS node.
/usr/bin/oc -n storage-project label nodes dhcp46-
9.lab.eng.blr.redhat.com storagenode=glusterfs 2>&1
node "dhcp46-9.lab.eng.blr.redhat.com" labeled
Marking 'dhcp46-134.lab.eng.blr.redhat.com' as a GlusterFS node.
/usr/bin/oc -n storage-project label nodes dhcp46-
134.lab.eng.blr.redhat.com storagenode=glusterfs 2>&1
node "dhcp46-134.lab.eng.blr.redhat.com" labeled
Deploying GlusterFS pods.
/usr/bin/oc -n storage-project process -p NODE_LABEL=glusterfs
glusterfs | /usr/bin/oc -n storage-project create -f - 2>&1
```

```
daemonset "glusterfs" created
Waiting for GlusterFS pods to start ...
Checking status of pods matching '--selector=glusterfs=pod':
glusterfs-6fj2v   1/1       Running   0          52s
glusterfs-ck40f   1/1       Running   0          52s
glusterfs-kbtz4   1/1       Running   0          52s
OK
/usr/bin/oc -n storage-project create secret generic heketi-
config-secret --from-file=private_key=/dev/null --from-
file=./heketi.json --from-file=topology.json=/opt/topology.json
secret "heketi-config-secret" created
/usr/bin/oc -n storage-project label --overwrite secret heketi-
config-secret glusterfs=heketi-config-secret heketi=config-secret
secret "heketi-config-secret" labeled
/usr/bin/oc -n storage-project process -p
HEKETI_EXECUTOR=kubernetes -p HEKETI_FSTAB=/var/lib/heketi/fstab
-p HEKETI_ADMIN_KEY= -p HEKETI_USER_KEY= deploy-heketi |
/usr/bin/oc -n storage-project create -f - 2>&1
service "deploy-heketi" created
route "deploy-heketi" created
deploymentconfig "deploy-heketi" created
Waiting for deploy-heketi pod to start ...
Checking status of pods matching '--selector=deploy-heketi=pod':
deploy-heketi-1-hf9rn   1/1       Running   0          2m
OK
Determining heketi service URL ... OK
/usr/bin/oc -n storage-project exec -it deploy-heketi-1-hf9rn --
heketi-cli -s http://localhost:8080 --user admin --secret ''
topology load --json=/etc/heketi/topology.json 2>&1
Creating cluster ... ID: 252509038eb8568162ec5920c12bc243
Allowing file volumes on cluster.
Allowing block volumes on cluster.
Creating node dhcp46-122.lab.eng.blr.redhat.com ... ID:
73ad287ae1ef231f8a0db46422367c9a
Adding device /dev/sdd ... OK
Adding device /dev/sde ... OK
Adding device /dev/sdf ... OK
Creating node dhcp46-9.lab.eng.blr.redhat.com ... ID:
0da1b20daaad2d5c57dbfc4f6ab78001
Adding device /dev/sdd ... OK
Adding device /dev/sde ... OK
Adding device /dev/sdf ... OK
Creating node dhcp46-134.lab.eng.blr.redhat.com ... ID:
4b3b62fc0efd298dedbcdacf0b498e65
Adding device /dev/sdd ... OK
Adding device /dev/sde ... OK
Adding device /dev/sdf ... OK
heketi topology loaded.
/usr/bin/oc -n storage-project exec -it deploy-heketi-1-hf9rn --
heketi-cli -s http://localhost:8080 --user admin --secret ''
setup-openshift-heketi-storage --listfile=/tmp/heketi-
storage.json --image rhgs3/rhgs-volmanager-rhel7:3.3.0-17 2>&1
Saving /tmp/heketi-storage.json
/usr/bin/oc -n storage-project exec -it deploy-heketi-1-hf9rn --
cat /tmp/heketi-storage.json | /usr/bin/oc -n storage-project
create -f - 2>&1
```

```
secret "heketi-storage-secret" created
endpoints "heketi-storage-endpoints" created
service "heketi-storage-endpoints" created
job "heketi-storage-copy-job" created

Checking status of pods matching '--selector=job-name=heketi-
storage-copy-job':
heketi-storage-copy-job-87v6n   0/1       Completed   0
7s
/usr/bin/oc -n storage-project label --overwrite svc heketi-
storage-endpoints glusterfs=heketi-storage-endpoints
heketi=storage-endpoints
service "heketi-storage-endpoints" labeled
/usr/bin/oc -n storage-project delete
all,service,jobs,deployment,secret --selector="deploy-heketi"
2>&1
deploymentconfig "deploy-heketi" deleted
route "deploy-heketi" deleted
service "deploy-heketi" deleted
job "heketi-storage-copy-job" deleted
pod "deploy-heketi-1-hf9rn" deleted
secret "heketi-storage-secret" deleted
/usr/bin/oc -n storage-project delete dc,route,template --
selector="deploy-heketi" 2>&1
template "deploy-heketi" deleted
/usr/bin/oc -n storage-project process -p
HEKETI_EXECUTOR=kubernetes -p HEKETI_FSTAB=/var/lib/heketi/fstab
-p HEKETI_ADMIN_KEY= -p HEKETI_USER_KEY= heketi | /usr/bin/oc -n
storage-project create -f - 2>&1
service "heketi" created
route "heketi" created
deploymentconfig "heketi" created
Waiting for heketi pod to start ...
Checking status of pods matching '--selector=heketi=pod':
heketi-1-zzblp   1/1       Running   0         31s
OK
Determining heketi service URL ... OK

heketi is now running and accessible via http://heketi-storage-
project.cloudapps.mystorage.com . To run
administrative commands you can install 'heketi-cli' and use it as
follows:

  # heketi-cli -s http://heketi-storage-
project.cloudapps.mystorage.com --user admin --secret
'<ADMIN_KEY>' cluster list

You can find it at https://github.com/heketi/heketi/releases .
Alternatively,
use it from within the heketi pod:

  # /usr/bin/oc -n storage-project exec -it <HEKETI_POD> --
heketi-cli -s http://localhost:8080 --user admin --secret
'<ADMIN_KEY>' cluster list

For dynamic provisioning, create a StorageClass similar to this:
```

```
---
apiVersion: storage.k8s.io/v1beta1
kind: StorageClass
metadata:
  name: glusterfs-storage
provisioner: kubernetes.io/glusterfs
parameters:
  resturl: "http://heketi-storage-
project.cloudapps.mystorage.com"

Ready to create and provide GlusterFS volumes.
sed -e 's/\${NAMESPACE}/storage-project/'
/usr/share/heketi/templates/glusterblock-provisioner.yaml |
/usr/bin/oc -n storage-project create -f - 2>&1
clusterrole "glusterblock-provisioner-runner" created
serviceaccount "glusterblock-provisioner" created
clusterrolebinding "glusterblock-provisioner" created
deploymentconfig "glusterblock-provisioner-dc" created
Waiting for glusterblock-provisioner pod to start ...
Checking status of pods matching '--selector=glusterfs=block-
provisioner-pod':
glusterblock-provisioner-dc-1-xm6bv   1/1       Running   0
6s
OK
Ready to create and provide Gluster block volumes.
/usr/bin/oc -n storage-project create secret generic heketi-
storage-project-admin-secret --from-literal=key= --
type=kubernetes.io/glusterfs
secret "heketi-storage-project-admin-secret" created
/usr/bin/oc -n storage-project label --overwrite secret heketi-
storage-project-admin-secret glusterfs=s3-heketi-storage-project-
admin-secret gluster-s3=heketi-storage-project-admin-secret
secret "heketi-storage-project-admin-secret" labeled
sed -e 's/\${STORAGE_CLASS}/glusterfs-for-s3/' -e
's/\${HEKETI_URL}/heketi-storage-
project.cloudapps.mystorage.com/' -e 's/\${NAMESPACE}/storage-
project/' /usr/share/heketi/templates/gluster-s3-
storageclass.yaml | /usr/bin/oc -n storage-project create -f -
2>&1
storageclass "glusterfs-for-s3" created
sed -e 's/\${STORAGE_CLASS}/glusterfs-for-s3/' -e
's/\${VOLUME_CAPACITY}/2Gi/' /usr/share/heketi/templates/gluster-
s3-pvcs.yaml | /usr/bin/oc -n storage-project create -f - 2>&1
persistentvolumeclaim "gluster-s3-claim" created
persistentvolumeclaim "gluster-s3-meta-claim" created

Checking status of persistentvolumeclaims matching '--
selector=glusterfs in (s3-pvc, s3-meta-pvc)':
gluster-s3-claim        Bound     pvc-35b6c1f0-9c65-11e7-9c8c-
005056b3ded1   2Gi       RWX       glusterfs-for-s3   18s
gluster-s3-meta-claim   Bound     pvc-35b86e7a-9c65-11e7-9c8c-
005056b3ded1   1Gi       RWX       glusterfs-for-s3   18s
/usr/bin/oc -n storage-project create -f
/usr/share/heketi/templates/gluster-s3-template.yaml 2>&1
template "gluster-s3" created
```

```
/usr/bin/oc -n storage-project process -p S3_ACCOUNT=testvolume -
p S3_USER=adminuser -p S3_PASSWORD=itsmine gluster-s3 |
/usr/bin/oc -n storage-project create -f - 2>&1
service "gluster-s3-service" created
route "gluster-s3-route" created
deploymentconfig "gluster-s3-dc" created
Waiting for gluster-s3 pod to start ...
Checking status of pods matching '--selector=glusterfs=s3-pod':
gluster-s3-dc-1-x3x4q   1/1       Running   0         6s
OK
Ready to create and provide Gluster object volumes.

Deployment complete!
```

3. Brick multiplexing is a feature that allows adding multiple bricks into one process. This reduces resource consumption, and allows us to run more bricks than before with the same memory consumption. Execute the following commands on one of the Red Hat Gluster Storage nodes on each cluster to enable brick-multiplexing:

    1. Execute the following command to enable brick multiplexing:

        ```
        # gluster vol set all cluster.brick-multiplex on
        ```

        For example:

        ```
        # gluster vol set all cluster.brick-multiplex on
        Brick-multiplexing is supported only for container workloads
        (CNS/CRS). Also it is advised to make sure that either all
        volumes are in stopped state or no bricks are running before this
        option is modified.Do you still want to continue? (y/n) y
        volume set: success
        ```

    2. Restart the heketidb volumes:

        ```
        # gluster vol stop heketidbstorage
        Stopping volume will make its data inaccessible. Do you want to
        continue? (y/n) y
        volume stop: heketidbstorage: success
        ```

        ```
        # gluster vol start heketidbstorage
        volume start: heketidbstorage: success
        ```

4. Execute the following command to let the client communicate with the container:

    ```
    # export  HEKETI_CLI_SERVER=http://heketi-<project_name>.
    <sub_domain_name>
    ```

    For example:

    ```
    # export  HEKETI_CLI_SERVER=http://heketi-storage-
    project.cloudapps.mystorage.com
    ```

    To verify if Heketi is loaded with the topology execute the following command:

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

NOTE

The cns-deploy tool does not support scaling up of the cluster. To manually scale-up the cluster, refer Chapter 12, *Managing Clusters*

Next step: If you are installing Container-Native Storage 3.9, proceed to Chapter 10, *Updating the Registry with Container-Native Storage as the Storage Back-end*.

CHAPTER 9. 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 9.1, "File Storage". Similarly, further details for provisioning volumes in block based storage is provided in Section 9.2, "Block Storage".

9.1. 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.

9.1.1. 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.

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:

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

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

5. 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.

6. 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.

lables: 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                 CAPACITY   ACCESSMODES   STATUS      CLAIM
REASON    AGE
glusterfs-4fc22ff9   100Gi      RWX           Available
4s
```

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

```
# oc get pvc

NAME              STATUS    VOLUME               CAPACITY
ACCESSMODES   AGE
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/openshift-container-platform/3.6/single/installation-and-configuration/#install-config-storage-examples-gluster-example.

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

```
# oc get pods
```

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/openshift-container-platform/3.6/single/installation-and-configuration/#gluster-volume-security.

9.1.2. Dynamic Provisioning of Volumes

Dynamic provisioning enables provisioning of Red Hat Gluster Storage volume to a running application container without having to pre-create 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.

NOTE

Dynamically provisioned Volumes are supported from Container-Native Storage 3.4. If you have any statically provisioned volumes and require more information about managing it, then refer Section 9.1.1, “Static Provisioning of Volumes”

9.1.2.1. 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.

9.1.2.1.1. 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
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 parameter 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 17, *Enabling Encryption*.

volumenameprefix: This is an optional parameter. It depicts the name of the volume created by heketi. For more information see Section 9.1.2.1.5, "(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 9.1.2.1.7, "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.
```

9.1.2.1.2. 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 Container-Native 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 Container-Native 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
```

9.1.2.1.3. 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:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 4Gi
```

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: RWO
   No events.
   ```

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

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

9.1.2.1.5. (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 refer, Section 9.1.2.1.1, “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:    RWO
    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.

9.1.2.1.6. 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/openshift-container-platform/3.6/single/installation-and-configuration/#install-config-storage-examples-gluster-example.

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

   ```
   # oc get pods

   NAME                                    READY     STATUS
   RESTARTS   AGE
   ```

```
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
```

9.1.2.1.7. 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 9.1.2.1.1, “Registering a Storage Class”

To expand the persistent value 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:
        PersistentVolumeClaimResize:
        configuration:
              apiVersion: v1
              disable: false
              kind: DefaultAdmissionConfig
   ```

   1. Restart the OpenShift master.

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% /etc/hosts
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.

9.1.2.1.8. Deleting a Persistent Volume Claim

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.
```

9.1.3. 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, refer https://access.redhat.com/documentation/en/openshift-container-platform/3.6/single/installation-and-configuration/#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 allows 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 9.1.2.1.3, “Creating a Persistent Volume Claim”

3. Use the claim in the pod. Ensure that this pod is non-privileged. For more information see, Section 9.1.2.1.6, “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.

9.1.4. 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.

To setup Prometheus, refer https://access.redhat.com/documentation/en-us/openshift_container_platform/3.9/html-single/installation_and_configuration/#openshift-prometheus.

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.

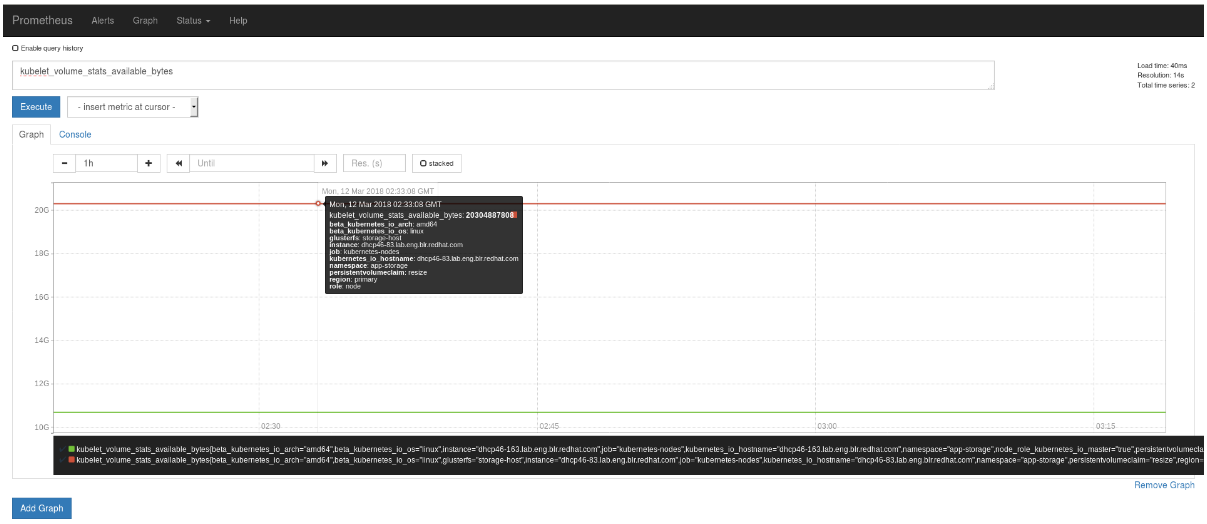
Viewing Metrics

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.)



kubelet_volume_stats_available_bytes

Load time: 40ms
Resolution: 14s
Total time series: 2

Mon, 12 Mar 2018 02:33:08 GMT

```
Mon, 12 Mar 2018 02:33:08 GMT
kubelet_volume_stats_available_bytes: 20304887808
beta_kubernetes_io_arch: amd64
beta_kubernetes_io_os: linux
glusterfs: storage-host
instance: dhcp46-83.lab.eng.blr.redhat.com
job: kubernetes-nodes
kubernetes_io_hostname: dhcp46-83.lab.eng.blr.redhat.com
namespace: app-storage
persistentvolumeclaim: resize
region: primary
role: node
```

```
✓ 🟩 kubelet_volume_stats_available_bytes{beta_kubernetes_io_arch="amd64",beta_kubernetes_io_os="linux",instance="dhcp46-163.lab.eng.blr.redhat.com",job="kubernetes-nodes",kubernetes_io_hostname="dhcp46-163.lab.eng.blr.redhat.com",namespace="app-storage",node_role_kubernetes_io_master="true",persistentvolumecla
✓ 🟥 kubelet_volume_stats_available_bytes{beta_kubernetes_io_arch="amd64",beta_kubernetes_io_os="linux",glusterfs="storage-host",instance="dhcp46-83.lab.eng.blr.redhat.com",job="kubernetes-nodes",kubernetes_io_hostname="dhcp46-83.lab.eng.blr.redhat.com",namespace="app-storage",persistentvolumeclaim="resize",region=
```

9.2. 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.

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

9.2.1. Dynamic Provisioning of Volumes for Block Storage

Dynamic provisioning enables provisioning of Red Hat Gluster Storage volume to a running application container without having to pre-create 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.

NOTE

If you are upgrading from Container-Native Storage 3.6 to Container-Native Storage 3.9, then ensure you refer Chapter 13, *Upgrading your Container-Native Storage Environment* before proceeding with the following steps.

9.2.1.1. 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.

9.2.1.1.1. 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
                   path_selector "round-robin 0"
                   failback immediate
                   path_checker "tur"
                   prio "const"
                   no_path_retry 120
                   rr_weight "uniform"
           }
   }
   EOF
   ```

4. Execute the following command to restart the multipath service:

```
# systemctl restart multipathd
```

9.2.1.1.2. 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 Container-Native 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
   ```

9.2.1.1.3. 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
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 9.2.1.1.6, “(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>
```

9.2.1.1.4. 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:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 5Gi
```

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:    RWO
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
```

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

NAME         STATUS    VOLUME
```

```
CAPACITY   ACCESSMODES   STORAGECLASS    AGE
pvc/claim1   Bound     pvc-ee30ff43-7ddc-11e7-89da-5254002ec671
5Gi        RWO           gluster-block   4m
```

9.2.1.1.6. (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 9.2.1.1.3, “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:    RWO
    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.

9.2.1.1.7. 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/openshift-container-platform/3.6/single/installation-and-configuration/#install-config-storage-examples-gluster-example.

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

```
# oc get pods

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
```

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
```

9.2.1.1.8. Deleting a Persistent Volume Claim

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 Container-Native Storage 3.9, and you want to use block storage as the backend storage for logging and metrics, proceed to Appendix E, *Gluster Block Storage as Backend for Logging and Metrics*.

CHAPTER 10. UPDATING THE REGISTRY WITH CONTAINER-NATIVE 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. Container-Native 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, refer to https://access.redhat.com/documentation/en-us/openshift_container_platform/3.9/html-single/installation_and_configuration/#setting-up-the-registry .

To include Container-Native Storage volume as a back-end when installing Openshift Container Platform, refer https://access.redhat.com/documentation/en-us/openshift_container_platform/3.9/html-single/installation_and_configuration/#advanced-install-containerized-glusterfs-backed-registry.

10.1. VALIDATING THE OPENSHIFT 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.
```

10.2. CONVERTING THE OPENSHIFT CONTAINER PLATFORM REGISTRY WITH CONTAINER-NATIVE 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-storage-endpoints -o yaml --
   namespace=storage-project > 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. Remove all the metadata except for **name**, 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-endpoints -o yaml --
namespace=storage-project > gluster-registry-service.yaml
```

7. Edit the **gluster-registry-service.yaml** file. Remove all the metadata except for name. 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. 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. Make the old registry readonly by executing the following command:

   ```
   # oc set env dc/docker-registry
   REGISTRY_STORAGE_MAINTENANCE_READONLY_ENABLED=true
   ```

2. 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
   ```

3. 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)}}')
   ```

4. Run rsync of data from old registry to the Red Hat Gluster Storage registry by executing the following command:

   ```
   # oc rsync $REGISTRY_POD:/registry/ $REGISTRY_POD:/gluster-
   registry/
   ```

5. Remove the Red Hat Gluster Storage registry form the old dc registry by executing the following command:

   ```
   # oc volume dc/docker-registry --remove --name=gluster-registry-
   storage
   ```

6.
   ```
   # oc volume dc/docker-registry --add --name=registry-storage -t
   pvc --claim-name=gluster-registry-claim --overwrite
   ```

7. Make the registry read write by executing the following command:

   ```
   # oc set env dc/docker-registry
   REGISTRY_STORAGE_MAINTENANCE_READONLY_ENABLED-
   ```

For more information about accessing the registry, see https://access.redhat.com/documentation/en-us/openshift_container_platform/3.9/html-single/installation_and_configuration/#install-config-registry-accessing.

Next step: If you are installing Container-Native Storage 3.9, you are now ready to create persistent volumes in one of the following formats.

- Section 9.1, “File Storage”: File 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.

- Section 9.2, “Block Storage”: Block storage allows each storage volume/block device to be treated as an independent disk drive, so that each storage volume/block device can support an individual file system..

CHAPTER 11. OPERATIONS ON A RED HAT GLUSTER STORAGE POD IN AN OPENSHIFT 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
   ```

   For example:

   ```
   # oc get pods
   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
   rhel1                                                    1/1
   Running   0          26s
   ```

   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>
   ```

   For example:

   ```
   # oc rsh glusterfs-dc-node1.example.com

   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_af42af87ad87ab4f01e8ca153abbbee9/brick
Brick6:
192.168.121.233:/var/lib/heketi/mounts/vg_5c6428c439eb6686c5e4cee565
32bacf/brick_ef41e04ca648efaf04178e64d25dbdcb/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
42af87ad87ab4f01e8ca153abbbee9/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
```

6. To use the snapshot feature, load the snapshot module using the following command:

```
# - modprobe dm_snapshot
```

IMPORTANT

Restrictions for using Snapshot

- After a snapshot is created, it must be accessed though 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.

7. 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, refer https://access.redhat.com/documentation/en-us/red_hat_gluster_storage/3.2/html-single/administration_guide/#chap-Managing_Snapshots.

10. You can set up Container-Native Storage volumes for geo-replication to a non-Container-Native Storage remote site. Geo-replication uses a master–slave model. Here, the Container-Native 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>
    ```

    For more information about setting up geo-replication, refer https://access.redhat.com/documentation/en-us/red_hat_gluster_storage/3.2/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.

CHAPTER 12. MANAGING CLUSTERS

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

12.1. 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.

12.1.1. 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

12.1.1.1. 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
```

12.1.1.2. Updating Topology File

You can add the new device to the node description in your topology JSON used to setup the cluster. Then rerun 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
```

12.1.2. 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 the API or by updating the topology JSON file. When you add a new node to the cluster, then you must register new devices to that node.

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#node_add

NOTE

Red Hat Gluster Storage pods have to be configured before proceeding with the following steps. To manually deploy the Red Hat Gluster Storage pods, refer Section A.2, “Deploying the Containers”

12.1.2.1. 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
```

12.1.2.2. 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
```

12.1.3. 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 Container-Native Storage
- Adding another Container-Native Storage cluster in a new project

12.1.3.1. Adding a New Cluster to the Existing Container-Native Storage

To add a new cluster to the existing Container-Native Storage, execute the following commands:

1. Verify that Container-Native 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     NODE-SELECTOR
   AGE
   glusterfs   3         3         3         storagenode=glusterfs
   8m
   ```

2. 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> storagenode=<node_label>
   ```

   where,

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

For example:

```
# oc label node 192.168.90.3 storagenode=glusterfs
node "192.168.90.3" labeled
```

3. Verify if the Red Hat Gluster Storage pods are running by executing the folowing command:

```
# oc get ds
```

For example:

```
# oc get ds
NAME        DESIRED   CURRENT   READY     NODE-SELECTOR
AGE
glusterfs   6         6         6         storagenode=glusterfs
8m
```

4. 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. A sample, 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.

5. 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
```

12.1.3.2. Adding Another Container-Native Storage Cluster in a New Project

To add another Container-Native 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 same label in two different projects. Node label is an argument to cns-deploy, thereby enabling deploying multiple trusted storage pool by using a different label in 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. A sample, 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: 63966f6ffd48c1980c4a2d03abeedd04
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, refer to the man page of the cns-deploy.

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

5. Verify that Container-Native 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
```

12.2. 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.

12.2.1. 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
```

12.2.2. 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.

12.2.2.1. 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
```

12.2.2.2. 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.

12.2.2.3. 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 **8faade64a9c8669de204b66bc083b10das**.

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
```

12.2.3. 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.

12.2.3.1. Disabling and Enabling a Node

Disabling node stops further allocation of bricks onto 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
```

12.2.3.2. 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

12.2.3.3. 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
   ...
   ...
   …
                   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 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
```

12.2.4. Deleting Clusters

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

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

For example:

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

CHAPTER 13. UPGRADING YOUR CONTAINER-NATIVE STORAGE ENVIRONMENT

This chapter describes the procedure to upgrade your environment from Container-Native Storage 3.6 to Container-Native Storage 3.9.

13.1. PREREQUISITES

Ensure the following prerequisites are met:

- Section 5.2.3, “Red Hat OpenShift Container Platform Requirements”
- Chapter 6, *Setting up Container-Native Storage*

13.2. UPGRADING CNS-DEPLOY AND HEKETI SERVER

The following commands must be executed on the client machine. If you want to set up a client machine, refer Section 5.2.1, “Installing Red Hat Gluster Storage Container Native with OpenShift Container Platform on Red Hat Enterprise Linux 7 based OpenShift Container Platform Cluster ” or Section 5.2.2, “Installing Red Hat Gluster Storage Container Native with OpenShift Container Platform on Red Hat Enterprise Linux Atomic Host OpenShift Container Platform Cluster”.

1. Execute the following command to update the heketi client and cns-deploy packages:

```
# yum update cns-deploy -y
# yum update heketi-client -y
```

2. Backup the Heketi database file

```
# oc rsh <heketi_pod_name>
# cp -a /var/lib/heketi/heketi.db /var/lib/heketi/heketi.db.`date
+%s`.`heketi --version | awk '{print $2}'`
# exit
```

3. Execute the following command to delete the heketi template

```
# oc delete templates heketi
```

4. Execute the following command to install the heketi template:

```
# oc create -f /usr/share/heketi/templates/heketi-template.yaml
template "heketi" created
```

5. Execute the following command to grant the heketi Service Account the neccessary privileges:

```
# oc policy add-role-to-user edit system:serviceaccount:
<project_name>:heketi-service-account
# oc adm policy add-scc-to-user privileged -z heketi-service-account
```

For example,

```
# oc policy add-role-to-user edit system:serviceaccount:storage-
project:heketi-service-account
# oc adm policy add-scc-to-user privileged -z heketi-service-account
```

6. Execute the following command to generate a new heketi configuration file:

   ```
   # sed -e "s/\${HEKETI_EXECUTOR}/kubernetes/" -e
   "s#\${HEKETI_FSTAB}#/var/lib/heketi/fstab#" -e "s/\${SSH_PORT}/22/"
   -e "s/\${SSH_USER}/root/" -e "s/\${SSH_SUDO}/false/" -e
   "s/\${BLOCK_HOST_CREATE}/true/" -e "s/\${BLOCK_HOST_SIZE}/500/"
   "/usr/share/heketi/templates/heketi.json.template" > heketi.json
   ```

   - The **`BLOCK_HOST_SIZE`** parameter controls the size (in GB) of the automatically created Red Hat Gluster Storage volumes hosting the gluster-block volumes (For more information, see Section 9.2, “Block Storage”). This default configuration will dynamically create block-hosting volumes of 500GB in size as more space is required.

   - Alternatively, copy the file **`/usr/share/heketi/templates/heketi.json.template`** to **`heketi.json`** in the current directory and edit the new file directly, replacing each "**`${VARIABLE}`**" string with the required parameter.

   NOTE

   JSON formatting is strictly required (e.g. no trailing spaces, booleans in all lowercase).

7. Execute the following command to create a secret to hold the configuration file:

   ```
   # oc create secret generic heketi-config-secret --from-
   file=heketi.json
   ```

8. Execute the following command to delete the deployment configuration, service, and route for heketi:

   ```
   # oc delete deploymentconfig,service,route heketi
   ```

9. Execute the following command to deploy the Heketi service which will be used to create persistent volumes for OpenShift:

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

   For example:

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

   service "heketi" created
   route "heketi" created
   deploymentconfig "heketi" created
   ```

10. Execute the following command to verify that the containers are running:

```
# oc get pods
```

For example:

```
# oc get pods
NAME                             READY     STATUS    RESTARTS
AGE
glusterfs-0h68l                  1/1       Running   0          3d
glusterfs-0vcf3                  1/1       Running   0          3d
glusterfs-gr9gh                  1/1       Running   0          3d
heketi-1-zpw4d                   1/1       Running   0          3h
storage-project-router-2-db2wl   1/1       Running   0          4d
```

13.3. UPGRADING THE RED HAT GLUSTER STORAGE PODS

The following commands must be executed on the client machine. If you want to set up a client machine, refer Section 5.2.1, “Installing Red Hat Gluster Storage Container Native with OpenShift Container Platform on Red Hat Enterprise Linux 7 based OpenShift Container Platform Cluster ” or Section 5.2.2, “Installing Red Hat Gluster Storage Container Native with OpenShift Container Platform on Red Hat Enterprise Linux Atomic Host OpenShift Container Platform Cluster”.

Following are the steps for updating a DaemonSet for glusterfs:

1. Execute the following command to find the DaemonSet name for gluster

   ```
   # oc get ds
   ```

2. Execute the following command to delete the DeamonSet:

   ```
   # oc delete ds <ds-name> --cascade=false
   ```

   Using **--cascade=false** option while deleting the old DaemonSet does not delete the gluster pods but deletes only the DaemonSet. After deleting the old DaemonSet, you must load the new one. When you manually delete the old pods, the new pods which are created will have the configurations of the new DaemonSet.

   For example,

   ```
   # oc delete ds glusterfs  --cascade=false
   daemonset "glusterfs" deleted
   ```

3. Execute the following commands to verify all the old pods are up:

   ```
   # oc get pods
   ```

   For example,

   ```
   # oc get pods
   NAME                             READY     STATUS    RESTARTS
   AGE
   glusterfs-0h68l                  1/1       Running   0          3d
   glusterfs-0vcf3                  1/1       Running   0          3d
   ```

```
glusterfs-gr9gh                  1/1       Running   0          3d
heketi-1-zpw4d                   1/1       Running   0          3h
storage-project-router-2-db2wl   1/1       Running   0          4d
```

4. Execute the following command to delete the old glusterfs template:

   ```
   # oc delete templates glusterfs
   ```

   For example,

   ```
   # oc delete templates glusterfs
   template “glusterfs” deleted
   ```

5. Label all the OpenShift Container Platform nodes that has the Red Hat Gluster Storage pods:

   1. Check if the nodes are labelled using the following command:

      ```
      # oc get nodes --show-labels
      ```

      If the Red Hat Gluster Storage nodes do not have the **`storagenode=glusterfs`** label, then proceed with the next step.

   2. Label all the OpenShift Container Platform nodes that has the Red Hat Gluster Storage pods:

      ```
      # oc label nodes <node name> storagenode=glusterfs
      ```

6. Execute the following command to register new gluster template:

   ```
   # oc create -f /usr/share/heketi/templates/glusterfs-template.yaml
   ```

   For example,

   ```
   # oc create -f /usr/share/heketi/templates/glusterfs-template.yaml
   template “glusterfs” created
   ```

7. Execute the following commands to start the gluster DeamonSet:

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

   For example,

   ```
   # oc process glusterfs | oc create -f -
   Deamonset “glusterfs” created
   ```

8. Execute the following command to identify the old gluster pods that needs to be deleted:

   ```
   # oc get pods
   ```

   For example,

```
# oc get pods
NAME                              READY     STATUS    RESTARTS
AGE
glusterfs-0h68l                   1/1       Running   0          3d
glusterfs-0vcf3                   1/1       Running   0          3d
glusterfs-gr9gh                   1/1       Running   0          3d
heketi-1-zpw4d                    1/1       Running   0          3h
storage-project-router-2-db2wl    1/1       Running   0          4d
```

9. Execute the following command to delete the old gluster pods. **`Gluster pods should follow rolling upgrade. Hence, you must ensure that the new pod is running before deleting the next old gluster pod. We support OnDelete Strategy DaemonSet update strategy`** . With **`OnDelete Strategy`** update strategy, after you update a DaemonSet template, new DaemonSet pods will only be created when you manually delete old DaemonSet pods.

    1. To delete the old gluster pods, execute the following command:

       ```
       # oc delete pod <gluster_pod>
       ```

       For example,

       ```
       # oc delete pod glusterfs-0vcf3
       pod  “glusterfs-0vcf3” deleted
       ```

       NOTE

       Before deleting the next pod, self heal check has to be made:

       1. Run the following command to access shell on gluster pod:

          ```
          # oc rsh <gluster_pod_name>
          ```

       2. Run the following command to obtain the volume names:

          ```
          # gluster volume list
          ```

       3. Run the following command on each volume to check the self-heal status:

          ```
          # gluster volume heal <volname> info
          ```

    2. The delete pod command will terminate the old pod and create a new pod. Run `# oc get pods -w` and check the **Age** of the pod and **READY** status should be 1/1. The following is the example output showing the status progression from termination to creation of the pod.

       ```
       # oc get pods -w
       NAME                             READY     STATUS
       RESTARTS   AGE
       glusterfs-0vcf3                  1/1       Terminating   0
       ```

```
3d
…

# oc get pods -w
NAME                             READY     STATUS
RESTARTS   AGE
glusterfs-pqfs6                  0/1       ContainerCreating   0
1s
…

# oc get pods -w
NAME                             READY     STATUS
RESTARTS   AGE
glusterfs-pqfs6                  1/1       Running       0
2m
```

10. Execute the following command to verify that the pods are running:

```
# oc get pods
```

For example,

```
# oc get pods
NAME                             READY     STATUS    RESTARTS
AGE
glusterfs-j241c                  1/1       Running   0          4m
glusterfs-pqfs6                  1/1       Running   0          7m
glusterfs-wrn6n                  1/1       Running   0
12m
heketi-1-zpw4d                   1/1       Running   0          4h
storage-project-router-2-db2wl   1/1       Running   0          4d
```

11. Execute the following command to verify if you have upgraded the pod to the latest version:

```
# oc rsh <gluster_pod_name> glusterd --version
```

For example:

```
 # oc rsh glusterfs-47qfc glusterd --version
glusterfs 3.8.4 built on Sep  6 2017 06:59:40
Repository revision: git://git.gluster.com/glusterfs.git
Copyright (c) 2006-2013 Red Hat, Inc. <http://www.redhat.com>
GlusterFS comes with ABSOLUTELY NO WARRANTY.
It is licensed to you under your choice of the GNU Lesser
General Public License, version 3 or any later version (LGPLv3
or later), or the GNU General Public License, version 2 (GPLv2),
in all cases as published by the Free Software Foundation.
```

12. Check the Red Hat Gluster Storage op-version by executing the following command:

```
# gluster vol get all cluster.op-version
```

   - Set the cluster.op-version to 31101 on any one of the pods:

NOTE

Ensure all the gluster pods are updated before changing the cluster.op-version.

```
# gluster volume set all cluster.op-version 31101
```

13. If a gluster-block-provisoner-pod already exists then delete it by executing the following commands:

    ```
    # oc delete pod <gluster-block-provisioner-pod>
    # oc delete dc <glsuter-block-dc>
    ```

    For example:

    ```
    # oc delete pod glusterblock-provisioner-dc-1-mmbbp
    # oc delete dc glusterblock-provisioner-dc
    ```

14. Execute the following commands to deploy the gluster-block provisioner:

    ```
    # sed -e 's/\\\${NAMESPACE}/<NAMESPACE>/'
    /usr/share/heketi/templates/glusterblock-provisioner.yaml | oc
    create -f -
    ```

    ```
    # oc adm policy add-cluster-role-to-user glusterblock-provisioner-
    runner system:serviceaccount:<NAMESPACE>:glusterblock-provisioner
    ```

    For example:

    ```
    # sed -e 's/\\\${NAMESPACE}/storage-project/'
    /usr/share/heketi/templates/glusterblock-provisioner.yaml | oc
    create -f -
    ```

    ```
    # oc adm policy add-cluster-role-to-user glusterblock-provisioner-
    runner system:serviceaccount:storage-project:glusterblock-
    provisioner
    ```

15. Brick multiplexing is a feature that allows adding multiple bricks into one process. This reduces resource consumption, and allows us to run more bricks than before with the same memory consumption. It is enabled by default from Container-Native Storage 3.6. During an upgrade from Container-Native Storage 3.6 to Container-Native Storage 3.9, to turn brick multiplexing on, execute the following commands:

    1. To exec into the Gluster pod, execute the following command and rsh into any of the gluster pods:

       ```
       # oc rsh <gluster_pod_name>
       ```

    2. Execute the following command to enable brick multiplexing:

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

For example:

```
# oc rsh glusterfs-770ql

sh-4.2# gluster volume set all cluster.brick-multiplex on
Brick-multiplexing is supported only for container workloads
(CNS/CRS). Also it is advised to make sure that either all
volumes are in stopped state or no bricks are running before this
option is modified.Do you still want to continue? (y/n) y
volume set: success
```

3. List all the volumes in the trusted storage pool:

   For example:

```
# gluster volume list

heketidbstorage
vol_194049d2565d2a4ad78ef0483e04711e
...
...
```

   Restart all the volumes:

```
# gluster vol stop <VOLNAME>
# gluster vol start <VOLNAME>
```

16. From Container-Native Storage 3.6, support for S3 compatible Object Store in Container-Native Storage is under technology preview. To enable S3 compatible object store, refer Chapter 18, *S3 Compatible Object Store in a Container-Native Storage Environment*.

CHAPTER 14. UPGRADING YOUR CONTAINER-READY STORAGE ENVIRONMENT

This chapter describes the procedures to follow to upgrade your Container-Ready Storage environment.

14.1. PREREQUISITES

Ensure the following prerequisites are met:

- Section 5.2.3, “Red Hat OpenShift Container Platform Requirements”
- Section 7.2, “Configuring Port Access”
- Section 7.3, “Enabling Kernel Modules”
- Section 7.4, “Starting and Enabling Services”
- If Heketi is running as a standalone service in one of the Red Hat Gluster Storage nodes, then ensure to open the port for Heketi. By default the port number for Heketi is 8080. To open this port execute the following command on the node where Heketi is running:

  ```
  # firewall-cmd --zone=zone_name --add-port=8080/tcp
  # firewall-cmd --zone=zone_name --add-port=8080/tcp --permanent
  ```

  If Heketi is configured to listen on a different port, then change the port number in the command accordingly.

14.2. UPGRADING CONTAINER-READY STORAGE

Follow the steps in the sections ahead to upgrade Container-Ready Storage.

14.2.1. Upgrading the Red Hat Gluster Storage Cluster

To upgrade the Red Hat Gluster Storage cluster, refer In-Service Software Upgrade.

14.2.2. Upgrading Heketi in RHGS node

NOTE

If Heketi is in an Openshift node, then skip this section and refer Section 14.2.3, “Upgrading Heketi in Openshift node” instead.

Upgrade Heketi by executing the following commands on the Red Hat Gluster Storage node where Heketi is running:

1. Backup the Heketi database file

   ```
   # cp -a /var/lib/heketi/heketi.db /var/lib/heketi/heketi.db.`date
   +%s`.`heketi --version | awk '{print $2}'`
   ```

2. Update Heketi by executing the following command in one of the Red Hat Gluster Storage nodes where Heketi is running:

```
# yum update heketi
```

3. Restart the Heketi service:

```
# systemctl restart heketi
```

14.2.3. Upgrading Heketi in Openshift node

The following commands must be executed on the client machine.

1. Execute the following command to update the heketi client and cns-deploy packages:

```
# yum update cns-deploy -y
# yum update heketi-client -y
```

2. Backup the Heketi database file

```
# oc rsh <heketi_pod_name>
# cp -a /var/lib/heketi/heketi.db /var/lib/heketi/heketi.db.`date
+%s`.`heketi --version | awk '{print $2}'`
# exit
```

3. Execute the following command to delete the heketi template

```
# oc delete templates heketi
```

4. In the heketi-template.yaml file localed at /usr/share/heketi/templates/heketi-template.yaml, change the value of HEKETI_EXECUTOR parameter from kuberenetes to ssh.

   For example:

```
parameters:
- name: HEKETI_USER_KEY
  displayName: Heketi User Secret
  description: Set secret for those creating volumes as type _user_
- name: HEKETI_ADMIN_KEY
  displayName: Heketi Administrator Secret
  description: Set secret for administration of the Heketi service
as user _admin_
- name: HEKETI_EXECUTOR
  displayName: heketi executor type
  description: Set the executor type, kubernetes or ssh
  value: ssh
- name: HEKETI_FSTAB
  displayName: heketi fstab path
  description: Set the fstab path, file that is populated with
bricks that heketi creates
  value: /var/lib/heketi/fstab
- name: HEKETI_IGNORE_STALE_OPERATIONS
  displayName: Whether to ignore stale operations at startup
  description: This allows to control whether heketi should start up
when there are stale pending operation entries present in the
database. Setting this to true lets heketi ignore existing pending
operations at startup. Setting it to false causes heketi to refuse
to start if pending operations are found in the database.
```

5. Execute the following command to install the heketi template:

```
# oc create -f /usr/share/heketi/templates/heketi-template.yaml
template "heketi" created
```

6. Execute the following command to grant the heketi Service Account the neccessary privileges:

```
# oc policy add-role-to-user edit system:serviceaccount:
<project_name>:heketi-service-account
# oc adm policy add-scc-to-user privileged -z heketi-service-account
```

For example,

```
# oc policy add-role-to-user edit system:serviceaccount:storage-
project:heketi-service-account
# oc adm policy add-scc-to-user privileged -z heketi-service-account
```

7. Execute the following command to generate a new heketi configuration file:

```
# sed -e "s/\${HEKETI_EXECUTOR}/ssh/" -e
"s#\${HEKETI_FSTAB}#/var/lib/heketi/fstab#" -e "s/\${SSH_PORT}/22/"
-e "s/\${SSH_USER}/root/" -e "s/\${SSH_SUDO}/false/" -e
"s/\${BLOCK_HOST_CREATE}/true/" -e "s/\${BLOCK_HOST_SIZE}/500/"
"/usr/share/heketi/templates/heketi.json.template" > heketi.json
```

- The **BLOCK_HOST_SIZE** parameter controls the size (in GB) of the automatically created Red Hat Gluster Storage volumes hosting the gluster-block volumes (For more information, see Section 9.2, “Block Storage”). This default configuration will dynamically create block-hosting volumes of 500GB in size as more space is required.

- Alternatively, copy the file **/usr/share/heketi/templates/heketi.json.template** to **heketi.json** in the current directory and edit the new file directly, replacing each "**${VARIABLE}**" string with the required parameter.

  NOTE

  JSON formatting is strictly required (e.g. no trailing spaces, booleans in all lowercase).

8. Execute the following command to create a secret to hold the configuration file:

```
# oc create secret generic heketi-config-secret --from-
file=private_key=${SSH_KEYFILE} --from-file=./heketi.json
```

9. Execute the following command to delete the deployment configuration, service, and route for heketi:

```
# oc delete deploymentconfig,service,route heketi
```

10. Execute the following command to deploy the Heketi service which will be used to create persistent volumes for OpenShift:

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

For example:

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

service "heketi" created
route "heketi" created
deploymentconfig "heketi" created
```

11. Execute the following command to verify that the containers are running:

```
# oc get pods
```

For example:

```
# oc get pods
NAME                              READY     STATUS    RESTARTS
AGE
glusterfs-0h68l                   1/1       Running   0          3d
glusterfs-0vcf3                   1/1       Running   0          3d
glusterfs-gr9gh                   1/1       Running   0          3d
heketi-1-zpw4d                    1/1       Running   0          3h
storage-project-router-2-db2wl    1/1       Running   0          4d
```

14.2.4. Upgrading Gluster Block

Execute the following steps to upgrade gluster block.

1. Execute the following command to install gluster block:

```
# yum install gluster-block
```

2. Enable and start the gluster block service:

```
# systemctl enable gluster-blockd
# systemctl start gluster-blockd
```

3. Execute the following command to update the heketi client and cns-deploy packages

```
# yum install cns-deploy -y
# yum update cns-deploy -y
# yum update heketi-client -y
```

4. To use gluster block, add the following two parameters to the **`glusterfs`** section in the heketi configuration file at /etc/heketi/heketi.JSON:

   ```
   auto_create_block_hosting_volume
   block_hosting_volume_size
   ```

   Where:

   `auto_create_block_hosting_volume`: Creates Block Hosting volumes automatically if not found or if the existing volume is exhausted. To enable this, set the value to **`true`**.

   `block_hosting_volume_size`: New block hosting volume will be created in the size mentioned. This is considered only if auto_create_block_hosting_volume is set to true. Recommended size is 500G.

   For example:

   ```
   .....
   .....
   "glusterfs" : {


                   "executor" : "ssh",

                   "db" : "/var/lib/heketi/heketi.db",

                   "sshexec" : {
                   "rebalance_on_expansion": true,
                   "keyfile" : "/etc/heketi/private_key"
                   },

                   "auto_create_block_hosting_volume": true,

                   "block_hosting_volume_size": 500G
           },
   .....
   .....
   ```

5. Restart the Heketi service:

   ```
   # systemctl restart heketi
   ```

6. If a gluster-block-provisoner-pod already exists then delete it by executing the following commands:

   ```
   # oc delete pod <gluster-block-provisioner-pod>
   # oc delete dc <glsuter-block-dc>
   ```

   For example:

   ```
   # oc delete pod glusterblock-provisioner-dc-1-mmbbp
   # oc delete dc glusterblock-provisioner-dc
   ```

7. Execute the following commands to deploy the gluster-block provisioner:

```
# sed -e 's/\\\${NAMESPACE}/<NAMESPACE>/'
/usr/share/heketi/templates/glusterblock-provisioner.yaml | oc
create -f -
```

```
# oc adm policy add-cluster-role-to-user glusterblock-provisioner-
runner system:serviceaccount:<NAMESPACE>:glusterblock-provisioner
```

For example:

```
# sed -e 's/\\\${NAMESPACE}/storage-project/'
/usr/share/heketi/templates/glusterblock-provisioner.yaml | oc
create -f -
```

```
# oc adm policy add-cluster-role-to-user glusterblock-provisioner-
runner system:serviceaccount:storage-project:glusterblock-
provisioner
```

14.2.5. Enabling S3 Compatible Object store

Support for S3 compatible Object Store is under technology preview. To enable S3 compatible object store, refer Chapter 18, *S3 Compatible Object Store in a Container-Native Storage Environment*.

CHAPTER 15. TROUBLESHOOTING

This chapter describes the most common troubleshooting scenarios related to Container-Native Storage.

- **What to do if a Container-Native Storage node Fails**

  If a Container-Native Storage node fails, and you want to delete it, then, disable the node before deleting it. For more information see, Section 12.2.3, "Deleting Node".

  If a Container-Native Storage node fails and you want to replace it, refer Section 12.2.3.3, "Replacing a Node".

- **What to do if a Container-Native Storage device fails**

  If a Container-Native Storage device fails, and you want to delete it, then, disable the device before deleting it. For more information see, Section 12.2.2, "Deleting Device " .

  If a Container-Native Storage device fails, and you want to replace it, refer Section 12.2.2.3, "Replacing a Device".

- **What to do if Container-Native 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 12.1, "Increasing Storage Capacity".

- **How to upgrade Openshift when Container-Native Storage is installed**

  To upgrade Openshift Container Platform refer, https://access.redhat.com/documentation/en-us/openshift_container_platform/3.9/html/installation_and_configuration/upgrading-a-cluster.

- **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 like 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**.

- **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 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 16. UNINSTALLING CONTAINERIZED RED HAT GLUSTER STORAGE

This chapter outlines the details for uninstalling containerized Red Hat Gluster Storage.

Perform the following steps for uninstalling:

1. **Cleanup Red Hat Gluster Storage using Heketi**

    1. Remove any containers using the persistent volume claim from Red Hat Gluster Storage.

    2. Remove the appropriate persistent volume claim and persistent volume:

        ```
        # oc delete pvc <pvc_name>
        # oc delete pv <pv_name>
        ```

2. **Remove all OpenShift objects**

    1. Delete all project specific pods, services, routes, and deployment configurations:

        ```
        # oc delete daemonset/glusterfs

        # oc delete deploymentconfig heketi

        # oc delete service heketi heketi-storage-endpoints

        # oc delete route heketi

        # oc delete endpoints heketi-storage-endpoints
        ```

        Wait until all the pods have been terminated.

    2. Check and delete the gluster service and endpoints from the projects that required a persistent storage:

        ```
        # oc get endpoints,service

        # oc delete endpoints <glusterfs-endpoint-name>

        # oc delete service <glusterfs-service-name>
        ```

3. **Cleanup the persistent directories**

    1. To cleanup the persistent directories execute the following command on each node as a root user:

        ```
        # rm -rf /var/lib/heketi \
           /etc/glusterfs \
           /var/lib/glusterd \
           /var/log/glusterfs
        ```

4. **Force cleanup the disks**

    1. Execute the following command to cleanup the disks:

```
# wipefs -a -f /dev/<disk-id>
```

CHAPTER 17. 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.

17.1. PREREQUISITES

To enable encryption it is necessary to have 3 certificates per node (glusterfs.key, gluserfs.pem and glusterfs.ca). For more information about the steps to be performed as prerequisites, refer https://access.redhat.com/documentation/en-us/red_hat_gluster_storage/3.2/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 Section 9.1.2.1.1, “Registering a Storage Class”.

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.

17.2. ENABLING ENCRYPTION FOR A NEW CONTAINER-NATIVE STORAGE SETUP

You can configure network encryption for a new Container-Native Storage setup for both I/O encryption and management encryption.

17.2.1. 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 17.2.2, “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, ie. 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 Container-Native Storage. For more information, see Section 8.2, “ Deploying Containerized Red Hat Gluster Storage Solutions”

17.2.2. 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 Container-Native Storage is deployed before proceeding with further steps. For more information see, Section 8.2, “ Deploying Containerized Red Hat Gluster Storage Solutions”

2. You can either create a statically provisioned volume or a dynamically provisioned volume. For more information about static provisioning of volumes, see Section 9.1.1, “Static Provisioning of Volumes” . For more information about dynamic provisioning of volumes, see Section 9.1.2, “Dynamic Provisioning of Volumes”

   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:   RWO
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 using the heketi-cli.

```
# 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
```

17.3. ENABLING ENCRYPTION FOR AN EXISTING CONTAINER-NATIVE STORAGE SETUP

You can configure network encryption for an existing Container-Native Storage setup for both I/O encryption and management encryption.

17.3.1. 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 using the heketi-cli.

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

17.3.2. 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 deamonset by executing the following 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.

17.4. DISABLING ENCRYPTION

You can disable encryption for on Container-Native Storage setup in the following two scenarios:

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

17.4.1. 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
```

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 deamonset by executing the following 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.

17.4.2. 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 deamonset by executing the following 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.

CHAPTER 18. S3 COMPATIBLE OBJECT STORE IN A CONTAINER-NATIVE 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.

18.1. PREREQUISITES

Before setting up S3 compatible object store for Container-Native Storage, ensure the following prerequisites are met:

- OpenShift setup must be up with master and nodes ready. For more information see Section 8.1, “Preparing the Red Hat OpenShift Container Platform Cluster”
- The cns-deploy tool is run and Heketi service is ready. For more information see, Section 8.2, “ Deploying Containerized Red Hat Gluster Storage Solutions”

18.2. SETTING UP S3 COMPATIBLE OBJECT STORE FOR CONTAINER-NATIVE STORAGE

Execute the following steps from the /usr/share/heketi/templates/ directory to set up S3 compatible object store for Container-Native 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 -
storageclass "gluster-s3-store" created
```

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 -
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>
```

18.3. 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.
- Update the s3curl.pl perl script with glusters3object url which was retreived:

  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/
  ```

APPENDIX A. MANUAL DEPLOYMENT

The following section covers the steps required to manually deploy Container-Native Storage.

A.1. INSTALLING THE TEMPLATES

Execute the following steps to register the Red Hat Gluster Storage and Heketi templates with OpenShift:

1. Use the newly created containerized Red Hat Gluster Storage project:

   ```
   # oc project project_name
   ```

   For example,

   ```
   # oc project storage-project
   Using project "storage-project" on server
   "https://master.example.com:8443".
   ```

2. Execute the following commands to install the templates:

   ```
   # oc create -f /usr/share/heketi/templates/deploy-heketi-
   template.yaml
   template "deploy-heketi" created
   ```

   ```
   # oc create -f /usr/share/heketi/templates/glusterfs-template.yaml
   template "glusterfs" created
   ```

   ```
   # oc create -f /usr/share/heketi/templates/heketi-service-
   account.yaml
   serviceaccount  "heketi-service-account" created
   ```

   ```
   # oc create -f /usr/share/heketi/templates/heketi-template.yaml
   template "heketi" created
   ```

3. Execute the following command to verify that the templates are installed:

   ```
   # oc get templates
   ```

   For example:

   ```
   # oc get templates

   NAME            DESCRIPTION                          PARAMETERS
   OBJECTS
   deploy-heketi   Bootstrap Heketi installation        2 (2 blank)   3
   glusterfs       GlusterFS DaemonSet template         0 (all set)
   1
   heketi          Heketi service deployment template   2 (2 blank)
   3
   ```

4. Execute the following command to verify that the serviceaccount is created:

```
# oc get serviceaccount heketi-service-account
```

For example:

```
# oc get serviceaccount heketi-service-account
NAME                     SECRETS   AGE
heketi-service-account   2         7d
```

A.2. DEPLOYING THE CONTAINERS

Execute the following commands to deploy the Red Hat Gluster Storage container on the nodes:

1. List out the hostnames of the nodes on which the Red Hat Gluster Storage container has to be deployed:

```
# oc get nodes
```

For example:

```
# oc get nodes

NAME                  STATUS                     AGE
node1.example.com     Ready                      12d
node2.example.com     Ready                      12d
node3.example.com     Ready                      12d
master.example.com    Ready,SchedulingDisabled   12d
```

2. Execute the following command to label all nodes that will run Red Hat Gluster Storage pods:

```
# oc label node <NODENAME> storagenode=glusterfs
```

For example:

```
# oc label nodes 192.168.90.3 storagenode=glusterfs
node "192.168.90.3" labeled
```

Repeat this command for every node that will be in the GlusterFS cluster.

Verify the label has set properly by running the following command:

```
# oc get nodes --show-labels
192.168.90.2   Ready                      12d
beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,kubernetes
.io/hostname=192.168.90.2,storagenode=glusterfs
192.168.90.3   Ready                      12d
beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,kubernetes
.io/hostname=192.168.90.3,storagenode=glusterfs
192.168.90.4   Ready                      12d
beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,kubernetes
.io/hostname=192.168.90.4,storagenode=glusterfs
```

```
192.168.90.5   Ready,SchedulingDisabled   12d
beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,kubernetes
.io/hostname=192.168.90.5
```

3. Execute the following command to deploy the Red Hat Gluster Storage pods:

```
# oc process glusterfs | oc create -f -
daemonset "glusterfs" created
```

NOTE

This does not initialize the hardware or create trusted storage pools. That aspect will be taken care by heketi which is explained in the further steps.

4. Execute the following command to grant the heketi Service Account the neccessary privileges:

```
# oc policy add-role-to-user edit system:serviceaccount:
<project_name>:heketi-service-account
# oc adm policy add-scc-to-user privileged -z heketi-service-account
```

For example:

```
# oc policy add-role-to-user edit system:serviceaccount:storage-
project:heketi-service-account
# oc adm policy add-scc-to-user privileged -z heketi-service-account
```

5. Execute the following command to deploy deploy-heketi:

```
# oc process deploy-heketi | oc create -f -
```

For example:

```
# oc process deploy-heketi | oc create -f -

service "deploy-heketi" created
route "deploy-heketi" created
deploymentconfig "deploy-heketi" created
```

6. Execute the following command to verify that the containers are running:

```
# oc get pods
```

For example:

```
# oc get pods
NAME                        READY     STATUS    RESTARTS   AGE
storage-project-router-1-pj9ea     1/1       Running   0
1d
deploy-heketi-1-m7x8g   1/1       Running   0          1m
glusterfs-41lfl         1/1       Running   0          1m
glusterfs-dtyr4         1/1       Running   0          1m
glusterfs-ral2d         1/1       Running   0          1m
```

A.3. SETTING UP THE HEKETI SERVER

After deploying the containers and installing the templates, the system is now ready to load the Heketi topology file. Heketi provides a RESTful management interface which can be used to manage the lifecycle of Red Hat Gluster Storage volumes.

A sample, formatted topology file (topology-sample.json) is installed with the ‘heketi-templates’ package in the **`/usr/share/heketi/`** directory.

```
{
    "clusters": [
               {
                  "nodes": [
                                {
                        "node": {
                            "hostnames": {
                                "manage": [
                                    "node1.example.com"
                                ],
                                "storage": [
                                    "192.168.121.168"
                                 ]
                             },
                             "zone": 1
                         },
                         "devices": [
                             "/dev/sdb",
                             "/dev/sdc",
                             "/dev/sdd",
                             "/dev/sde"
                         ]
               },...
```

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.

IMPORTANT

Heketi stores its database on a Red Hat Gluster Storage volume. Heketi service does not respond if the volume is down.

To resolve this issue, restart the gluster pods hosting the Heketi volume.

Execute the following steps to set up the Heketi server:

1. Execute the following command to check if the bootstrap container is running:

   ```
   # curl http://deploy-heketi-<project_name>.<sub-domain_name>/hello
   ```

   For example:

```
# curl http://deploy-heketi-storage-
project.cloudapps.mystorage.com/hello
Hello from Heketi
```

2. Execute the following command to load the topology file:

   ```
   # export  HEKETI_CLI_SERVER=http://deploy-heketi-<project_name>.
   <sub_domain_name>
   ```

   For example:

   ```
   # export HEKETI_CLI_SERVER=http://deploy-heketi-storage-
   project.cloudapps.mystorage.com
   ```

   ```
   # heketi-cli topology load --json=topology.json
   ```

   For example:

   ```
   # heketi-cli topology load --json=topology.json
   Creating cluster ... ID: 94877b3f72b79273e87c1e94201ecd58
    Creating node node1.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 node2.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 node3.example.com ... ID:
   73fe4aa89ba35c51de4a51ecbf52544d
           Adding device /dev/sdb ... OK
           Adding device /dev/sdc ... OK
           Adding device /dev/sdd ... OK
           Adding device /dev/sde ... OK
   ```

3. Execute the following command to verify that the topology is loaded:

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

4. Execute the following command to create the Heketi storage volume which will store the database on a reliable Red Hat Gluster Storage volume:

   ```
   # heketi-cli setup-openshift-heketi-storage
   ```

   For example:

   ```
   # heketi-cli setup-openshift-heketi-storage
   Saving heketi-storage.json
   ```

5. Execute the following command to create a job which will copy the database from deploy-heketi bootstrap container to the volume.

   ```
   # oc create -f heketi-storage.json
   ```

   For example:

   ```
   # oc create -f heketi-storage.json
   secret "heketi-storage-secret" created
   endpoints "heketi-storage-endpoints" created
   service "heketi-storage-endpoints" created
   job "heketi-storage-copy-job" created
   ```

6. Execute the following command to verify that the job has finished successfully:

   ```
   # oc get jobs
   ```

   For example:

   ```
   # oc get jobs
   NAME                      DESIRED   SUCCESSFUL   AGE
   heketi-storage-copy-job   1            1               2m
   ```

7. Execute the following command to remove all resources used to bootstrap heketi:

   ```
   # oc delete all,job,template,secret --selector="deploy-heketi"
   ```

   For example:

   ```
   # oc delete all,job,template,secret --selector="deploy-heketi"
   deploymentconfig "deploy-heketi" deleted
   route "deploy-heketi" deleted
   service "deploy-heketi" deleted
   pod "deploy-heketi-1-4k1fh" deleted
   job "heketi-storage-copy-job" deleted
   template "deploy-heketi" deleted
   ```

8. Execute the following command to deploy the Heketi service which will be used to create persistent volumes for OpenShift:

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

   For example:

   ```
   # oc process heketi | oc create -f -
   service "heketi" created
   route "heketi" created
   deploymentconfig "heketi" created
   ```

9. Execute the following command to let the client communicate with the container:

```
# export  HEKETI_CLI_SERVER=http://heketi-<project_name>.
<sub_domain_name>
```

For example:

```
# export  HEKETI_CLI_SERVER=http://heketi-storage-
project.cloudapps.mystorage.com
```

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

APPENDIX B. 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 adm policy add-cluster-role-to-user cluster-admin admin
   ```

For more information on authentication methods, see https://access.redhat.com/documentation/en-us/openshift_container_platform/3.9/html-single/installation_and_configuration/#identity-providers.

APPENDIX C. 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 D. HEKETI CLI COMMANDS

This section provides a list of some of the useful heketi-cli commands:

- **heketi-cli topology info**

  This command retreives information about the current Topology.

- **heketi-cli cluster list**

  Lists the clusters managed by Heketi

  For example:

  ```
  # heketi-cli cluster list
  Clusters:
  9460bbea6f6b1e4d833ae803816122c6
  ```

- **heketi-cli cluster info <cluster_id>**

  Retrieves the information about the cluster.

  For example:

  ```
  # heketi-cli cluster info 9460bbea6f6b1e4d833ae803816122c6
  Cluster id: 9460bbea6f6b1e4d833ae803816122c6
  Nodes:
  1030f9361cff8c6bfde7b9b079327c78
  30f2ab4d971da572b03cfe33a1ba525f
  f648e1ddc0b95f3069bd2e14c7e34475
  Volumes:
  142e0ec4a4c1d1cc082071329a0911c6
  638d0dc6b1c85f5eaf13bd5c7ed2ee2a
  ```

- **heketi-cli node info <node_id>**

  Retrieves the information about the node.

  For example:

  ```
  # heketi-cli node info 1030f9361cff8c6bfde7b9b079327c78
  Node Id: 1030f9361cff8c6bfde7b9b079327c78
  State: online
  Cluster Id: 9460bbea6f6b1e4d833ae803816122c6
  Zone: 1
  Management Hostname: node1.example.com
  Storage Hostname: 10.70.41.202
  Devices:
  Id:69214867a4d32251aaf1dcd77cb7f359   Name:/dev/vdg
  State:online    Size (GiB):4999    Used (GiB):253     Free
  (GiB):4746
  Id:6cd437c304979ea004abc2c4da8bdaf4   Name:/dev/vde
  State:online    Size (GiB):4999    Used (GiB):354     Free
  (GiB):4645
  Id:d2e9fcd9da04999ddab11cab651e18d2   Name:/dev/vdf
  State:online    Size (GiB):4999    Used (GiB):831     Free
  (GiB):4168
  ```

_

- **heketi-cli volume list**

  Lists the volumes managed by Heketi

  For example:

  ```
  # heketi-cli volume list
  Id:142e0ec4a4c1d1cc082071329a0911c6
  Cluster:9460bbea6f6b1e4d833ae803816122c6    Name:heketidbstorage
  Id:638d0dc6b1c85f5eaf13bd5c7ed2ee2a
  Cluster:9460bbea6f6b1e4d833ae803816122c6    Name:scalevol-1
  ```

For more information, refer to the man page of the heketi-cli.

```
# heketi-cli --help
```

The command line program for Heketi.

Usage

- heketi-cli [flags]
- heketi-cli [command]

For example:

```
# export HEKETI_CLI_SERVER=http://localhost:8080
```

```
# heketi-cli volume list
```

The available commands are listed below:

- **cluster**

  Heketi cluster management

- **device**

  Heketi device management

- **setup-openshift-heketi-storage**

  Setup OpenShift/Kubernetes persistent storage for Heketi

- **node**

  Heketi Node Management

- **topology**

  Heketi Topology Management

- **volume**

  Heketi Volume Management

APPENDIX E. 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.

E.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
    ```

E.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/3.9/html-single/installation_and_configuration/#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_es_pvc_dynamic=true
openshift_logging_es_pvc_size=5G
openshift_logging_es_cluster_size=3
openshift_logging_es_memory_limit=4G
openshift_logging_es_number_of_replicas=2
openshift_logging_es_nodeselector={'region' : 'infra'}
openshift_logging_curator_nodeselector={'region' : 'infra'}
openshift_logging_kibana_nodeselector={'region' : 'infra'}
```

3. Run the Ansible playbook. For more information, see .https://access.redhat.com/documentation/en-us/openshift_container_platform/3.9/html-single/installation_and_configuration/#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/3.9/html-single/installation_and_configuration/#install-config-aggregate-logging-sizing-guidelines-storage.

E.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/3.9/html-single/installation_and_configuration/#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_cassandra_storage_type=dynamic
openshift_metrics_cassandra_pvc_size=5G
openshift_metrics_cassandra_replicas=3
openshift_metrics_cassandra_limits_memory=2G
```

```
openshift_metrics_cassandra_nodeselector={'region':'infra'}
openshift_metrics_hawkular_nodeselector={'region':'infra'}
openshift_metrics_heapster_nodeselector={'region':'infra'}
```

3. Run the Ansible playbook. For more information, see https://access.redhat.com/documentation/en-us/openshift_container_platform/3.9/html-single/installation_and_configuration/#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/3.9/html-single/installation_and_configuration/#metrics-data-storage.

E.4. VERIFYING IF GLUSTER BLOCK IS SETUP AS BACKEND

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

1. To get an overiew 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 refer, https://access.redhat.com/documentation/en-us/openshift_container_platform/3.9/html/installation_and_configuration/install-config-aggregate-logging-sizing.

APPENDIX F. REVISION HISTORY

Revision 1.0-03	**Thu Apr 26 2018**	**Bhavana Mohan**

Updated the OCP version 3.7 in the Supported Versions section. Publishing the guide for CNS 3.9 Async release.

Revision 1.0-02	**Thu Apr 05 2018**	**Bhavana Mohan**

Publishing the guide for CNS 3.9 release.

Revision 1.0-01	**Thu Apr 05 2018**	**Bhavana Mohan**

Added statement (and links) informing customers of preferred OCP ansible installation workflow.

Documented support for storage class with heketi to enable use of encryption in flight/rest .

Documented how to specify a custom name for a persistent volume using the volnameprefix parameter in the storageclass file.

Documented how to expand a persistent volume.

Documented how to view details and metrics that show the persistent volume consumption.

Documented steps (and workflow matrix) regarding how to upgrade from CRS 3.6 to CRS 3.9 in both an OCP 3.7 environment and an OCP 3.9 environment.

Documented how to install CNS 3.9 with OCP.

Documented how to install CRS 3.9 with OCP.

Documented tasks on how to upgrade to CNS 3.9

Added "Next step" information in the deployment workflow.