Oracle Cloud Native Environment

Storage for Release 1.6

F75657-04
December 2023

Oracle Cloud Native Environment Storage for Release 1.6,

F75657-04

Contents

Preface

This document contains information about setting up and using persistent storage in Oracle Cloud Native Environment. It describes the modules provided with Oracle Cloud Native Environment to set up persistent storage.

Documentation License

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at https://www.oracle.com/corporate/accessibility/.

For information about the accessibility of the Oracle Help Center, see the Oracle Accessibility Conformance Report at https://www.oracle.com/corporate/accessibility/templates/t2-11535.html.

Access to Oracle Support for Accessibility

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit https://www.oracle.com/corporate/accessibility/learning-support.html#support-tab.

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

1

Introduction to Storage

Every meaningful workload in the computing industry requires some sort of data storage. Persistent storage is essential when working with stateful applications like databases, as it is important that you are able to retain data beyond the lifecycle of the container, or even of the pod itself.

Persistent storage in Kubernetes is handled in the form of PersistentVolume objects and are bound to pods using a PersistentVolumeClaim. You can host a PersistentVolume locally or on networked storage devices or services.

A typical Kubernetes environment involves multiple hosts and usually includes some type of networked storage. Using networked storage helps to ensure resilience and allows you to take full advantage of a clustered environment. In the case where the node where a pod is running fails, a new pod can be started on an alternate node and storage access can be resumed. This is particularly important for database environments where replica setup has been properly configured.

Persistent Storage Concepts

Persistent storage is provided in Kubernetes using the PersistentVolume subsystem. To configure persistent storage, you should be familiar with the following terms:

- **PersistentVolume**

  A PersistentVolume defines the type of storage that is being used and the method used to connect to it. This is the real disk or networked storage service that is used to store data.

- **PersistentVolumeClaim**

  A PersistentVolumeClaim defines the parameters that a consumer, like a pod, uses to bind the PersistentVolume. The claim may specify quota and access modes that should be applied to the resource for a consumer. A pod can use a PersistentVolumeClaim to gain access to the volume and mount it.

- **StorageClass**

  A StorageClass is an object that specifies a volume plug-in, known as a provisioner, that allows users to define PersistentVolumeClaims without needing to preconfigure the storage for a PersistentVolume. This can be used to provide access to similar volume types as a pooled resource that can be dynamically provisioned for the lifecycle of a PersistentVolumeClaim.

PersistentVolumes can be provisioned either statically or dynamically.

Static PersistentVolumes are manually created and contain the details required to access real storage and can be consumed directly by any pod that has an associated PersistentVolumeClaim.

Dynamic PersistentVolumes can be automatically generated if a PersistentVolumeClaim does not match an existing static PersistentVolume and an existing StorageClass is requested in the claim. A StorageClass can be defined to host a pool of storage that can be accessed

dynamically. Creating a StorageClass is an optional step that is only required if you intend to use dynamic provisioning.

The process to provision persistent storage is as follows:

1. Create a PersistentVolume or StorageClass.
2. Create PersistentVolumeClaims.
3. Configure a pod to use the PersistentVolumeClaim.

The process for adding and configuring NFS and iSCSI volumes is described in detail in the upstream documentation at:

https://kubernetes.io/docs/concepts/storage/persistent-volumes/

Container Storage Interface Plug-ins

The Container Storage Interface (CSI) is an Open Container Initiative standard for controlling storage workloads from container engines. Kubernetes implements this interface to provide automated control for storage workloads inside Kubernetes clusters. For a list of the Kubernetes storage provisioners, see the upstream documentation at:

https://kubernetes.io/docs/concepts/storage/storage-classes/

You can install CSI plug-ins into a Kubernetes cluster in Oracle Cloud Native Environment. To make it easier to perform the CSI plug-in installation, Oracle provides a number of storage related modules.

The Oracle Cloud Infrastructure Cloud Controller Manager module for Oracle Cloud Native Environment can be used to set up the CSI plug-in for Oracle Cloud Infrastructure.

The Gluster Container Storage Interface module for Oracle Cloud Native Environment can be used to set up the CSI plug-in for Glusterfs.

More information on these modules is included in this guide.

Introduction to the Oracle Cloud Infrastructure Cloud Controller Manager Module

Oracle Cloud Infrastructure block volumes provide reliable, high-performance block storage designed to work with a range of virtual machines and bare metal instances. With built-in redundancy, block volumes are persistent and durable beyond the lifespan of a virtual machine and can scale to 1 PB per compute instance.

The Oracle Cloud Infrastructure Cloud Controller Manager module is used to set up dynamically provisioned persistent storage using Oracle Cloud Infrastructure block volumes.

The Oracle Cloud Infrastructure Cloud Controller Manager module creates a Kubernetes StorageClass provisioner to access storage on Oracle Cloud Infrastructure block volumes. The Kubernetes Cloud Controller Manager (`oci-cloud-controller-manager`) is a CSI plug-in for Kubernetes clusters running on Oracle Cloud Infrastructure. The Kubernetes Cloud Controller Manager is used to dynamically provision Oracle Cloud Infrastructure volumes for use as Kubernetes

PersistentVolumes. The Platform API Server communicates with the Oracle Cloud Infrastructure API to provision and manage Oracle Cloud Infrastructure volumes using PersistentVolumeClaims. The Oracle Cloud Infrastructure volumes can be automatically destroyed when the PersistentVolumeClaims are deleted.

For more information on the Kubernetes Cloud Controller Manager, see the upstream documentation at:

https://github.com/oracle/oci-cloud-controller-manager

Introduction to the Gluster Container Storage Interface Module

> **Important:**
>
> The Gluster Container Storage Interface module, used to install Gluster and set up Glusterfs, is deprecated. The Gluster Container Storage Interface module may be removed in a future release.

Gluster is a scalable, distributed file system that aggregates disk storage resources from multiple servers into a single global namespace. Heketi is used to create and manage volumes in a Gluster cluster.

The Gluster Container Storage Interface module is used to set up dynamically provisioned persistent storage using Gluster Storage for Oracle Linux.

Oracle Cloud Native Environment does not deploy Gluster or Heketi. Gluster Storage for Oracle Linux and the Heketi API must be installed and configured separately, before it can be added to Oracle Cloud Native Environment.

The Gluster Container Storage Interface module creates a Kubernetes StorageClass provisioner to access existing storage on Glusterfs. Kubernetes uses the Glusterfs plug-in to dynamically provision Gluster volumes for use as Kubernetes PersistentVolumes. The Platform API Server communicates with the Heketi API to provision and manage Gluster volumes using PersistentVolumeClaims. The Gluster volumes can be automatically destroyed when the PersistentVolumeClaims are deleted.

2
Using Oracle Cloud Infrastructure Storage

This chapter discusses how to install and use the Oracle Cloud Infrastructure Cloud Controller Manager module to set up dynamically provisioned persistent storage for Kubernetes applications in Oracle Cloud Native Environment on Oracle Cloud Infrastructure instances.

Prerequisites

Before you set up the Oracle Cloud Infrastructure Cloud Controller Manager module, you need to gather information about your Oracle Cloud Infrastructure environment. The most common information you need is:

- The identifier for the region.
- The OCID for the tenancy.
- The OCID for the compartment.
- The OCID for the user.
- The public key fingerprint for the API signing key pair.
- The private key file for the API signing key pair.

You may need more information related to your Oracle Cloud Infrastructure networking or other components.

For information on finding each of these identifiers or components, see the Oracle Cloud Infrastructure documentation.

Deploying the Oracle Cloud Infrastructure Cloud Controller Manager Module

If you have already installed the Oracle Cloud Infrastructure Cloud Controller Manager module to make use of Oracle Cloud Infrastructure application load balancers, you do not need to create another module to provision storage. The Oracle Cloud Infrastructure Cloud Controller Manager module is used to provision both Oracle Cloud Infrastructure storage and load balancers.

You can deploy all the modules required to set up Oracle Cloud Infrastructure storage for a Kubernetes cluster using a single `olcnectl module create` command. This method might be useful if you want to deploy the Oracle Cloud Infrastructure Cloud Controller Manager module at the same time as deploying a Kubernetes cluster.

If you have an existing deployment of the Kubernetes module, you can specify that instance when deploying the Oracle Cloud Infrastructure Cloud Controller Manager module.

This section guides you through installing each component required to deploy the Oracle Cloud Infrastructure Cloud Controller Manager module.

For the full list of the Platform CLI command options available when creating modules, see the `olcnectl module create` command in Platform Command-Line Interface.

To deploy the Oracle Cloud Infrastructure Cloud Controller Manager module:

1. If you do not already have an environment set up, create one into which the modules can be deployed. For information on setting up an environment, see Getting Started. The name of the environment in this example is `myenvironment`.
2. If you do not already have a Kubernetes module set up or deployed, set one up.

   For information on adding a Kubernetes module to an environment, see Container Orchestration. The name of the Kubernetes module in this example is `mycluster`.
3. Create an Oracle Cloud Infrastructure Cloud Controller Manager module and associate it with the Kubernetes module named `mycluster` using the `--oci-ccm-kubernetes-module` option. In this example, the Oracle Cloud Infrastructure Cloud Controller Manager module is named `myoci`.

   > **Important:**
   >
   > The following example uses the `--oci-private-key-file` option that is to be used in Release 1.6.1 onwards. For Release 1.6.0, use `--oci-private-key` option instead.

   ```
   olcnectl module create \
   --environment-name myenvironment \
   --module oci-ccm \
   --name myoci \
   --oci-ccm-kubernetes-module mycluster \
   --oci-region us-ashburn-1 \
   --oci-tenancy ocid1.tenancy.oc1..unique_ID \
   --oci-compartment ocid1.compartment.oc1..unique_ID \
   --oci-user ocid1.user.oc1..unique_ID \
   --oci-fingerprint b5:52:... \
   --oci-private-key-file /home/opc/.oci/oci_api_key.pem
   ```

   The `--module` option sets the module type to create, which is `oci-ccm`. You define the name of the Oracle Cloud Infrastructure Cloud Controller Manager module using the `--name` option, which in this case is `myoci`.

   The `--oci-ccm-kubernetes-module` option sets the name of the Kubernetes module.

   The `--oci-region` option sets the Oracle Cloud Infrastructure region to use. The region in this example is `us-ashburn-1`.

   The `--oci-tenancy` option sets the OCID for your tenancy.

   The `--oci-compartment` option sets the OCID for your compartment.

   The `--oci-user` option sets the OCID for the user.

   The `--oci-fingerprint` option sets the fingerprint for the public key for the Oracle Cloud Infrastructure API signing key.

The `--oci-private-key-file` path option sets the location of the private key for the Oracle Cloud Infrastructure API signing key. This must be located on the operator node.

If you do not include all the required options when adding the module, you are prompted to provide them.

4. Use the `olcnectl module validate` command to validate the Oracle Cloud Infrastructure Cloud Controller Manager module can be deployed to the nodes. For example:

```
olcnectl module validate \
--environment-name myenvironment \
--name myoci
```

5. Use the `olcnectl module install` command to install the Oracle Cloud Infrastructure Cloud Controller Manager module. For example:

```
olcnectl module install \
--environment-name myenvironment \
--name myoci
```

The Oracle Cloud Infrastructure Cloud Controller Manager module is deployed into the Kubernetes cluster.

Verifying the Oracle Cloud Infrastructure Cloud Controller Manager Deployment

You can verify the Oracle Cloud Infrastructure Cloud Controller Manager module is deployed using the `olcnectl module instances` command on the operator node. For example:

```
olcnectl module instances \
--environment-name myenvironment
INSTANCE                 MODULE        STATE
mycluster                kubernetes    installed
myoci                    oci-ccm       installed
control1.example.com     node          installed
...
```

Note the entry for `oci-ccm` in the `MODULE` column is in the `installed` state.

In addition, use the `olcnectl module report` command to review information about the module. For example, use the following command to review the Oracle Cloud Infrastructure Cloud Controller Manager module named `myoci` in `myenvironment`:

```
olcnectl module report \
--environment-name myenvironment \
--name myoci \
--children
```

For more information on the syntax for the `olcnectl module report` command, see Platform Command-Line Interface.

On a control plane node, you can also verify the `oci-bv` StorageClass for the Oracle Cloud Infrastructure provisioner is created using the `kubectl get sc` command:

```
kubectl get sc
NAME     PROVISIONER                       RECLAIMPOLICY   VOLUMEBINDINGMODE      ...
oci-bv   blockvolume.csi.oraclecloud.com   Delete          WaitForFirstConsumer   ...
```

You can get more details about the StorageClass using the `kubectl describe sc` command. For example:

```
kubectl describe sc oci-bv
Name:                  oci-bv
IsDefaultClass:        No
Annotations:           meta.helm.sh/release-name=myoci,meta.helm.sh/release-
namespace=default
Provisioner:           blockvolume.csi.oraclecloud.com
Parameters:            <none>
AllowVolumeExpansion:  <unset>
MountOptions:          <none>
ReclaimPolicy:         Delete
VolumeBindingMode:     WaitForFirstConsumer
Events:                <none>
```

Creating Oracle Cloud Infrastructure Block Storage

This section contains a basic test to verify you can create Oracle Cloud Infrastructure block storage to provide persistent storage to applications running on Kubernetes.

To create a test application to use Oracle Cloud Infrastructure storage:

1. Create a Kubernetes PersistentVolumeClaim file. On a control plane node, create a file named `pvc.yaml`. Copy the following into the file.

   ```
   apiVersion: v1
   kind: PersistentVolumeClaim
   metadata:
    name: myoci-pvc
   spec:
    accessModes:
     - ReadWriteOnce
    storageClassName: oci-bv
    resources:
     requests:
       storage: 50Gi
   ```

   Note that the `accessModes` setting for Oracle Cloud Infrastructure storage must be `ReadWriteOnce`. The minimum Oracle Cloud Infrastructure block size is 50Gi.

2. Create the Kubernetes PersistentVolumeClaim.

   ```
   kubectl apply -f pvc.yaml
   persistentvolumeclaim/myoci-pvc created
   ```

3. You can see the PersistentVolumeClaim is created using the `kubectl get pvc` command:

   ```
   kubectl get pvc
   NAME        STATUS    VOLUME   CAPACITY   ACCESS MODES   STORAGECLASS   AGE
   myoci-pvc   Pending                                      oci-bv         15s
   ```

   The `STATUS` is `Pending` and means the claim is waiting for an application to claim it.

   You can get more details about the PersistentVolumeClaim using the `kubectl describe pvc` command. For example:

   ```
   kubectl describe pvc myoci-pvc
   Name:          myoci-pvc
   Namespace:     default
   ```

```
StorageClass:  oci-bv
Status:        Pending
Volume:
Labels:        <none>
Annotations:   <none>
Finalizers:    [kubernetes.io/pvc-protection]
Capacity:
Access Modes:
VolumeMode:    Filesystem
Used By:       <none>
Events:
  Type    Reason                Age
From                         ...
  ----    ------                ----
----
  Normal  WaitForFirstConsumer  2m18s (x26 over 8m29s)  persistentvolume-
controller  ...
```

4. Create a Kubernetes application that uses the PersistentVolumeClaim. Create a file named `nginx.yaml` and copy the following into the file.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    run: mynginx
  name: mynginx
spec:
  replicas: 1
  selector:
    matchLabels:
      run: mynginx
  template:
    metadata:
      labels:
        run: mynginx
    spec:
      containers:
      - image: container-registry.oracle.com/olcne/nginx:1.17.7
        name: mynginx
        ports:
        - containerPort: 80
        volumeMounts:
        - name: nginx-pvc
          mountPath: /usr/share/nginx/html
      volumes:
      - name: nginx-pvc
        persistentVolumeClaim:
          claimName: myoci-pvc
```

5. Start the application:

```
kubectl apply -f nginx.yaml
deployment.apps/mynginx created
```

6. You can see the application is running using the `kubectl get deployment` command:

```
kubectl get deployment
NAME      READY   UP-TO-DATE   AVAILABLE   AGE
mynginx   1/1     1            1           63s
```

7. You can see the application is using the PersistentVolumeClaim to provide persistent storage on Oracle Cloud Infrastructure using the `kubectl describe deployment` command:

```
kubectl describe deployment mynginx
...
Pod Template:
  Labels:  run=mynginx
  Containers:
   mynginx:
    Image:        container-registry.oracle.com/olcne/nginx:1.17.7
    Port:         80/TCP
    Host Port:    0/TCP
    Environment:  <none>
    Mounts:
      /usr/share/nginx/html from nginx-pvc (rw)
  Volumes:
   nginx-pvc:
    Type:       PersistentVolumeClaim (a reference to a
PersistentVolumeClaim in the same namespace)
    ClaimName:  myoci-pvc
    ReadOnly:   false
...
```

Note the `ClaimName` is `myoci-pvc`, which is the name of the PersistentVolumeClaim created earlier.

You can see the PersistentVolumeClaim is now bound to this application using the `kubectl get pvc` command:

```
kubectl get pvc
NAME        STATUS   VOLUME             CAPACITY   ACCESS MODES
STORAGECLASS   AGE
myoci-pvc   Bound    csi-84175067-...   50Gi       RWO            oci-
bv         1m
```

> **Tip:**
>
> If you log in to Oracle Cloud Infrastructure, you can see there is a block volume created with the name listed in the `VOLUME` column. The block volume is attached to the compute instance on which the Kubernetes application is running.

8. You can delete the test application using:

```
kubectl delete deployment mynginx
deployment.apps "mynginx" deleted
```

9. You can delete the PersistentVolumeClaim using:

```
kubectl delete pvc myoci-pvc
persistentvolumeclaim "myoci-pvc" deleted
```

The storage is deleted.

> **Tip:**
>
> If you log in to Oracle Cloud Infrastructure, you can see the block volume is terminated.

Removing the Oracle Cloud Infrastructure Cloud Controller Manager Module

You can remove a deployment of the Oracle Cloud Infrastructure Cloud Controller Manager module and leave the Kubernetes cluster in place. To do this, you remove the Oracle Cloud Infrastructure Cloud Controller Manager module from the environment.

Use the `olcnectl module uninstall` command to remove the Oracle Cloud Infrastructure Cloud Controller Manager module. For example, to uninstall the Oracle Cloud Infrastructure Cloud Controller Manager module named `myoci` in the environment named `myenvironment`:

```
olcnectl module uninstall \
--environment-name myenvironment \
--name myoci
```

The Oracle Cloud Infrastructure Cloud Controller Manager module is removed from the environment.

3
Using Gluster Storage

This chapter discusses how to install and use the Gluster Container Storage Interface module to set up dynamically provisioned persistent storage for Kubernetes applications using Gluster Storage for Oracle Linux and Heketi in Oracle Cloud Native Environment.

> **Important:**
>
> The Gluster Container Storage Interface module, used to install Gluster and set up Glusterfs, is deprecated. The Gluster Container Storage Interface module may be removed in a future release.

Prerequisites

You need to have a Gluster Storage for Oracle Linux cluster set up and ready to use. You must also install Heketi in the Gluster cluster. The Platform API Server communicates with the Heketi API to provision and manage Gluster volumes.

You do not need to create any Gluster volumes as these are dynamically provisioned as required.

The basic requirements for setting up Gluster are:

- Install Gluster on each node in the Gluster cluster.
- Set up the cluster to access volumes using the Gluster native client (FUSE) method.
- Install Heketi and create the Gluster cluster.
- Make sure you can connect to the Heketi API from the operator node.

For information on installing and setting up Gluster Storage for Oracle Linux and Heketi, see Oracle® Linux: Gluster Storage for Oracle Linux User's Guide.

Deploying the Gluster Module

You can deploy all the modules required to set up Gluster storage for a Kubernetes cluster using a single `olcnectl module create` command. This method might be useful if you want to deploy the Gluster Container Storage Interface module at the same time as deploying a Kubernetes cluster.

If you have an existing deployment of the Kubernetes module, you can specify that instance when deploying the Gluster Container Storage Interface module.

This section guides you through installing each component required to deploy the Gluster Container Storage Interface module.

For the full list of the Platform CLI command options available when creating modules, see the `olcnectl module create` command in Platform Command-Line Interface.

To deploy the Gluster Container Storage Interface module:

1. If you do not already have an environment set up, create one into which the modules can be deployed. For information on setting up an environment, see Getting Started. The name of the environment in this example is `myenvironment`.
2. If you do not already have a Kubernetes module set up or deployed, set one up. For information on adding a Kubernetes module to an environment, see Container Orchestration. The name of the Kubernetes module in this example is `mycluster`.
3. Create a Gluster Container Storage Interface module and associate it with the Kubernetes module named `mycluster` using the `--gluster-kubernetes-module` option. In this example, the Gluster Container Storage Interface module is named `mygluster`.

   ```
   olcnectl module create \
   --environment-name myenvironment \
   --module gluster \
   --name mygluster \
   --gluster-kubernetes-module mycluster \
   --gluster-server-url https:\\mygluster.example.com:8080
   ```

   The `--module` option sets the module type to create, which is `gluster`. You define the name of the Gluster Container Storage Interface module using the `--name` option, which in this case is `mygluster`.

   The `--gluster-kubernetes-module` option sets the name of the Kubernetes module.

   The `--gluster-server-url` option sets the location of the Heketi API server, which in this example is `https:\\mygluster.example.com:8080`. You do not need to include this option if Heketi is on the operator node and using HTTP, as the default for this option is `http://127.0.0.1:8080`.

   > **Tip:**
   >
   > Make sure you can reach the Heketi API from the operator node using `curl`, for example:
   >
   > ```
   > curl -w "\n" https:\\mygluster.example.com:8080/hello
   > ```
   >
   > Or if Heketi is on the operator node using HTTP:
   >
   > ```
   > curl -w "\n" http:\\127.0.0.1:8080/hello
   > ```
   >
   > You should see returned:
   >
   > ```
   > Hello from Heketi.
   > ```

   If you do not include all the required options when adding the module, you are prompted to provide them.

   There are some optional command options that you may need to include if you are not using the default values, such as `--gluster-server-user` and `--gluster-secret-key`.
4. Use the `olcnectl module validate` command to validate the Gluster Container Storage Interface module can be deployed to the nodes. For example:

```
olcnectl module validate \
--environment-name myenvironment \
--name mygluster
```

5. Use the `olcnectl module install` command to install the Gluster Container Storage Interface module. For example:

```
olcnectl module install \
--environment-name myenvironment \
--name mygluster
```

The Gluster Container Storage Interface module is deployed into the Kubernetes cluster.

Verifying the Gluster Module Deployment

You can verify the Gluster Container Storage Interface module is deployed using the `olcnectl module instances` command on the operator node. For example:

```
olcnectl module instances \
--environment-name myenvironment
INSTANCE                 MODULE        STATE
mycluster                kubernetes    installed
mygluster                gluster       installed
control1.example.com     node          installed
...
```

Note the entry for `gluster` in the `MODULE` column is in the `installed` state.

In addition, use the `olcnectl module report` command to review information about the module. For example, use the following command to review the Gluster Container Storage Interface module named `mygluster` in `myenvironment`:

```
olcnectl module report \
--environment-name myenvironment \
--name mygluster \
--children
```

For more information on the syntax for the `olcnectl module report` command, see Platform Command-Line Interface.

On a control plane node, you can also verify the StorageClass for the Glusterfs provisioner is created using the `kubectl get sc` command:

```
kubectl get sc
NAME                       PROVISIONER              RECLAIMPOLICY
VOLUMEBINDINGMODE   ...
hyperconverged (default)   kubernetes.io/glusterfs   Delete
Immediate           ...
```

In this case, the StorageClass is named `hyperconverged`, which is the default name.

You can get more details about the StorageClass using the `kubectl describe sc` command. For example:

```
kubectl describe sc hyperconverged
Name:                  hyperconverged
IsDefaultClass:        Yes
Annotations:           meta.helm.sh/release-name=mygluster,meta.helm.sh/release-
namespace=defau...
Provisioner:           kubernetes.io/glusterfs
```

```
Parameters:
restauthenabled=true,resturl=http://...:8080,restuser=admin,secretName=a...
AllowVolumeExpansion:  <unset>
MountOptions:          <none>
ReclaimPolicy:         Delete
VolumeBindingMode:     Immediate
Events:                <none>
```

Creating a Gluster Volume

This section contains a basic test to verify you can create a Gluster volume to provide persistent storage to applications running on Kubernetes.

To create a test application to use Glusterfs:

1. Create a Kubernetes PersistentVolumeClaim file. On a control plane node, create a file named `pvc.yaml`. Copy the following into the file.

   ```
   apiVersion: v1
   kind: PersistentVolumeClaim
   metadata:
    name: mygluster-pvc
   spec:
    accessModes:
     - ReadWriteMany
    resources:
     requests:
       storage: 1Gi
   ```

2. Create the Kubernetes PersistentVolumeClaim.

   ```
   kubectl apply -f pvc.yaml
   persistentvolumeclaim/mygluster-pvc created
   ```

3. You can see the PersistentVolumeClaim is created using the `kubectl get pvc` command:

   ```
   kubectl get pvc
   NAME           STATUS    VOLUME         CAPACITY   ACCESS MODES
   STORAGECLASS     AGE
   mygluster-pvc  Bound     pvc-59f70...   1Gi        RWX
   hyperconverged   18s
   ```

   You can get more details about the PersistentVolumeClaim using the `kubectl describe pvc` command. For example:

   ```
   kubectl describe pvc mygluster-pvc
   Name:          mygluster-pvc
   Namespace:     default
   StorageClass:  hyperconverged
   Status:        Bound
   Volume:        pvc-59f7047b-9287-4163-9cff-c669cfbd4970
   Labels:        <none>
   Annotations:   pv.kubernetes.io/bind-completed: yes
                  pv.kubernetes.io/bound-by-controller: yes
                  volume.beta.kubernetes.io/storage-provisioner: kubernetes.io/
   glusterfs
   Finalizers:    [kubernetes.io/pvc-protection]
   Capacity:      1Gi
   Access Modes:  RWX
   VolumeMode:    Filesystem
   ```

```
Used By:       <none>
Events:
  Type    Reason                 Age   From                           Message
  ----    ------                 ----  ----                           -------
  Normal  ProvisioningSucceeded  73s   persistentvolume-controller  Successfully
provi...
```

4. Create a Kubernetes application that uses the PersistentVolumeClaim. Create a file named `nginx.yaml` and copy the following into the file.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    run: mynginx
  name: mynginx
spec:
  replicas: 1
  selector:
    matchLabels:
      run: mynginx
  template:
    metadata:
      labels:
        run: mynginx
    spec:
      containers:
      - image: container-registry.oracle.com/olcne/nginx:1.17.7
        name: mynginx
        ports:
        - containerPort: 80
        volumeMounts:
        - name: nginx-pvc
          mountPath: /usr/share/nginx/html
      volumes:
      - name: nginx-pvc
        persistentVolumeClaim:
          claimName: mygluster-pvc
```

5. Start the application:

```
kubectl apply -f nginx.yaml
deployment.apps/mynginx created
```

6. You can see the application is running using the `kubectl get deployment` command:

```
kubectl get deployment
NAME      READY   UP-TO-DATE   AVAILABLE   AGE
mynginx   1/1     1            1           16s
```

7. You can see the application is using the PersistentVolumeClaim to provide persistent storage on Glusterfs using the `kubectl describe deployment` command:

```
kubectl describe deployment mynginx
...
Pod Template:
  Labels:  run=mynginx
  Containers:
   mynginx:
    Image:        container-registry.oracle.com/olcne/nginx:1.17.7
    Port:         80/TCP
    Host Port:    0/TCP
    Environment:  <none>
```

```
    Mounts:
      /usr/share/nginx/html from nginx-pvc (rw)
  Volumes:
   nginx-pvc:
    Type:       PersistentVolumeClaim (a reference to a
PersistentVolumeClaim in the ...
    ClaimName:  mygluster-pvc
    ReadOnly:   false
```

8. You can delete the test application using:

```
kubectl delete deployment mynginx
deployment.apps "mynginx" deleted
```

9. You can delete the PersistentVolumeClaim using:

```
kubectl delete pvc mygluster-pvc
persistentvolumeclaim "mygluster-pvc" deleted
```

Removing the Gluster Module

You can remove a deployment of the Gluster Container Storage Interface module and leave the Kubernetes cluster in place. To do this, you remove the Gluster Container Storage Interface module from the environment.

Use the `olcnectl module uninstall` command to remove the Gluster Container Storage Interface module. For example, to uninstall the Gluster Container Storage Interface module named `mygluster` in the environment named `myenvironment`:

```
olcnectl module uninstall \
--environment-name myenvironment \
--name mygluster
```

The Gluster Container Storage Interface module is removed from the environment.