Persistent Storage for Kubernetes using Hedvig

This document talks about how Hedvig integrates natively with Kubernetes as a scale-out distributed storage platform for stateful containerized applications. Specifically, Hedvig integrates with the Kubernetes Persistent Volume framework to allow end users to manage all aspects of persistent volumes using native Kubernetes constructs.

Persistent Volume Framework

Before presenting the Hedvig-Kubernetes integration, it is necessary to understand how Kubernetes manages the lifecycle of storage resources. In a nutshell,

- A storage resource is configured by a `StorageClass`
- A storage resource is provisioned as a `PersistentVolume`
- A storage resource is consumed through a `PersistentVolumeClaim`

StorageClass, PersistentVolume and PersistentVolumeClaim are the three main constructs described by the Persistent Volume framework.

PersistentVolume resources are used to manage durable storage in a cluster. PersistentVolumes can be dynamically provisioned; the user does not have to manually create and delete the backing storage.

PersistentVolumes are cluster resources that exist independently of Pods. This means that the disk and data represented by a PersistentVolume continue to exist as the cluster changes and as Pods are deleted and recreated.

A *PersistentVolumeClaim* is a request for and claim to a PersistentVolume resource. PersistentVolumeClaim objects request a specific size, access mode, and StorageClass for the PersistentVolume. If a PersistentVolume that satisfies the request exists, or can be provisioned, the PersistentVolumeClaim is bound to that PersistentVolume.

Pods use claims as Volumes. The cluster inspects the claim to find the bound PersistentVolume and mounts that PersistentVolume for the Pod.

A *StorageClass* provides a way for administrators to describe the “classes” of storage that

2350 Mission College Blvd, Suite 500
Santa Clara, CA 95054
www.hedvig.io

they offer. Different classes might map to quality-of-service levels, or to backup policies, or to arbitrary policies determined by cluster administrators.

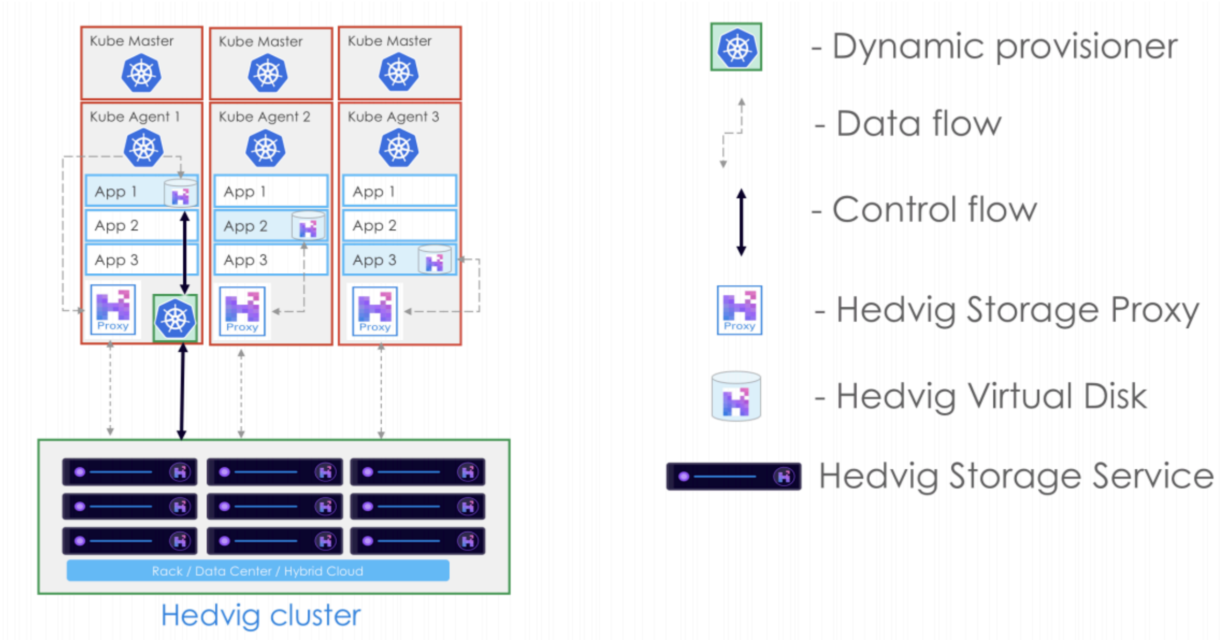
Dynamic Provisioning

Dynamic volume provisioning allows storage volumes to be created on-demand. Before dynamic provisioning, cluster administrators had to manually make calls to their storage provider to provision new storage volumes, and then create Persistent Volume (PV) objects to represent them in Kubernetes.

With dynamic provisioning, these two steps are automated, eliminating the need for cluster administrators to pre-provision storage. Instead, storage resources can be dynamically provisioned using the provisioner specified by the StorageClass.

Architecture

The following figure provides an overview of how Hedvig integrates with any Kubernetes cluster, outlining the different components and how they interact with each other.



Hedvig Dynamic Provisioner is an out-of-tree storage provisioner for Hedvig that allow Kubernetes users to provision Hedvig virtual disks and consume them as persistent

2350 Mission College Blvd, Suite 500
Santa Clara, CA 95054
www.hedvig.io

volumes using native Kubernetes constructs. Hedvig Dynamic Provisioner operates by setting a watch at the Kubernetes API server for events (add/update/delete) specific to Persistent Volume constructs. Hedvig Dynamic Provisioner is installed as a deployment in Kubernetes.

Hedvig Storage Proxy is Hedvig's iSCSI target that enables Kubernetes users to consume Hedvig virtual disks as iSCSI persistent volumes for stateful applications. Hedvig Storage Proxy enables client-side caching and deduplication with local SSD and PCIe flash resources for fast local reads and efficient data transfers. It also provides an encryption engine for data in-flight and at rest.

Hedvig Storage Proxy is deployed as a Daemonset in the Kubernetes cluster. This ensures that Kubernetes spawns one Hedvig Storage Proxy pod on every Kubernetes node, thereby enabling applications to migrate between Kubernetes nodes without losing access to their volumes or data. As the Kubernetes cluster scales, Hedvig Storage Proxy scales with it automatically.

Storage Operations

The following steps describe a sequence of events that occur when a Kubernetes user issues a request to provision storage. These events explain how the Hedvig components interact with Kubernetes and utilize the Kubernetes constructs to allow end users to seamlessly manage Hedvig storage within a Kubernetes cluster.

1. The administrator creates one or more storage classes (**StorageClass**) for Hedvig.
2. The user creates a **PersistentVolumeClaim** by specifying the **StorageClass** to use and the size of **PersistentVolume** requested.
3. Kubernetes identifies **Hedvig Dynamic Provisioner** as the provisioner to use for this **PersistentVolumeClaim** based on the **StorageClass** specified.
4. **Hedvig Dynamic Provisioner** provisions a **Hedvig Virtual Disk** on the underlying Hedvig cluster with the size requested and the attributes listed in the **StorageClass**.
5. **Hedvig Dynamic Provisioner** presents the virtual disk as a LUN to the Hedvig Storage Proxies and creates a **PersistentVolume** in Kubernetes of type iSCSI

2350 Mission College Blvd, Suite 500
Santa Clara, CA 95054
www.hedvig.io

corresponding to the Hedvig virtual disk.

6. Kubernetes binds the **PersistentVolumeClaim** to the **PersistentVolume** created. The **PersistentVolume** can be consumed by any pod that uses the **PersistentVolumeClaim**.

Installation

This section will walk you through the process of installing the Hedvig Dynamic Provisioner.

Download the latest version of the Hedvig Dynamic Provisioner installer tarball from the Hedvig portal onto any client machine configured to run the `kubectl` commands.

```
# tar -xvzf hedvig-installer.tar.gz
# ls hedvig-installer/
hedvig-clusterrolebindings-k8s.yaml  hedvigctl             hedvig-namespace
.yaml        install_hedvig.sh  setup

hedvig-clusterroles-k8s.yaml         hedvig-deployment.yaml  hedvig-serviceac
counts.yaml  manifests          uninstall_hedvig.sh
```

Update the following configuration values in the setup/backend.json file to point to the Hedvig Storage Cluster: * `StorageCluster` — Name of the Hedvig Storage Cluster * `StorageNode` — Hostname/IP address of one of the Hedvig Storage Cluster Nodes * `KubeClusterID` — Unique id for the Kubernetes cluster

Update the image name in hedvig-deployment.yaml to `hedviginc/hedvigprovisioner:<tag>` and set the `<tag>` to the most recently released version of the Hedvig Dynamic Provisioner.

A complete list of available versions can be found here:
https://hub.docker.com/r/hedviginc/hedvigprovisioner/tags/

Hedvig

2350 Mission College Blvd, Suite 500
Santa Clara, CA 95054
www.hedvig.io

Install the Hedvig Dynamic Provisioner (in the hedvig namespace) by running the following command -

```
# ./install_hedvig.sh -n hedvig
Installer assumes you have deployed Kubernetes. If this is an OpenShift
deployment, make sure 'oc'
is in the $PATH.

serviceaccount/hedvig created
clusterrole.rbac.authorization.k8s.io/hedvig created
clusterrolebinding.rbac.authorization.k8s.io/hedvig created
configmap/hedvig-launcher-config created
deployment.extensions/hedvig created
Hedvig deployment definition is available in /root/hedvig-installer/hedvig-de
ployment.yaml.
Started Hedvig Provisioner in namespace "hedvig".

+----------------------+----------------+--------+---------+
|         NAME         | STORAGE DRIVER | ONLINE | VOLUMES |
+----------------------+----------------+--------+---------+
| hedvig-block-backend | hedvig-block   | true   |       0 |
+----------------------+----------------+--------+---------+
Create the backend in namespace "hedvig".
```

Note: Hedvig Dynamic Provisioner only supports "hedvig-block-backend" today. Therefore, all the persistent volumes provisioned will have their access modes set to "ReadWriteOnce".

Using the Hedvig Dynamic Provisioner

This section describes the Kubernetes workflows involved in provisioning and managing persistent volumes.

2350 Mission College Blvd, Suite 500
Santa Clara, CA 95054
www.hedvig.io

Add a storage class

A StorageClass can be created by providing a unique name for the storage class and specifying the provisioner to be used for the storage class. The following manifest creates a default StorageClass for Hedvig:

```
apiVersion: storage.k8s.io/v1beta1
kind: StorageClass
metadata:
  name: sc-hedvig-default
provisioner: hedvig.io/provisioner
parameters:
  backendType: "hedvig-block"
```

In addition to this, storage classes can be customized by providing Hedvig virtual disk attributes as parameters. The following manifest creates a StorageClass with compression and deduplication enables for persistent volumes:

```
apiVersion: storage.k8s.io/v1beta1
kind: StorageClass

metadata:
  name: sc-hedvig-compressed-dedup
provisioner: hedvig.io/provisioner
parameters:
  backendType: "hedvig-block"
  compressed: "true"
  dedupEnable: "true"
```


2350 Mission College Blvd, Suite 500
Santa Clara, CA 95054
www.hedvig.io

The set of all the Hedvig virtual disk parameters are listed in the following table –

Key	Values	Default Value	Notes
dedupEnable	true/false	false	
compressed	true/false	false	
cacheEnable	true/false	false	
rf	1 to 6	3	
rp	Agnostic/RackAware/ DataCenterAware	Agnostic	
dcNames	comma-separated list of data center names		This applies only to a replication policy (rp) of DataCenterAware
diskResidence	flash/hdd	hdd	In an all-flash cluster, diskResidence should always be set to flash
encryptionEnable	true/false	false	
blockSize	512/4096	4096	
description	any string		

Provision a volume

A persistent volume is be dynamically provisioned on Hedvig by creating a PersistentVolumeClaim. The following manifest creates a PersistentVolumeClaim with the StorageClass created in the previous section:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:

  name: pvc-centos-test
  annotations:
    volume.beta.kubernetes.io/storage-class: sc-hedvig-default
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
```

This results in the creation of a PersistentVolume (and a corresponding Hedvig virtual disk), which is bound to the PersistentVolumeClaim `pvc-centos-test`.

```
# kubectl get pvc
NAME            STATUS    VOLUME                          CAPACITY   ACCESS MODES
STORAGECLASS       AGE
pvc-centos-test   Bound    default-pvc-centos-test-c7b00   10Gi      RWO          sc-hedvig-default
8h

# kubectl get pv
NAME                            CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS
 CLAIM                  STORAGECLASS       REASON    AGE
default-pvc-centos-test-c7b00   10Gi       RWO           Delete          Bound     default/pvc-centos-
test   sc-hedvig-default            8h
```

The default reclaim policy for the dynamically created persistent volumes is set to “Delete”. If the PersistentVolumeClaim is deleted, the PersistentVolume bound to it (and the corresponding Hedvig virtual disk) are also deleted.

In order to retain a PersistentVolume beyond the lifetime of its PersistentVolumeClaim, set the reclaim policy to “Retain” as shown below.

2350 Mission College Blvd, Suite 500
Santa Clara, CA 95054
www.hedvig.io

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-centos-test
  annotations:
    volume.beta.kubernetes.io/storage-class: sc-hedvig-default
    provisioner.hedvig.io/reclaimPolicy: Retain
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
```

Choose a filesystem

The default filesystem type for the dynamically created persistent volumes is set to "xfs". This can be changed by using an annotation in the PersistentVolumeClaim. Kubernetes currently supports the following filesystems - ext2/3/4 and xfs.

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-centos-test
  annotations:
    volume.beta.kubernetes.io/storage-class: sc-hedvig-default
    provisioner.hedvig.io/reclaimPolicy: Retain
    provisioner.hedvig.io/fileSystem: ext4
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
```


2350 Mission College Blvd, Suite 500
Santa Clara, CA 95054
www.hedvig.io

Kubernetes added support for custom mount options for certain native PersistentVolume types (for e.g. iSCSI) in version 1.9 through the PersistentVolumeSpec.

Mount options must be specified in the storage class and any PersistentVolume created using that storage class will be mounted using the corresponding mount options.

The following manifest describes a StorageClass with custom mount options:

```
apiVersion: storage.k8s.io/v1beta1
kind: StorageClass
metadata:
  name: sc-hedvig-custom-mount
provisioner: hedvig.io/provisioner
mountOptions: ["noatime","nodiratime","max_batch_time=0"]
parameters:
  backendType: "hedvig-block"
```

Caveats while using mount options * Kubernetes does not validate the mount options specified. Therefore, if the mount options specified do not apply to the filesystem type chosen for the PV, the mount call will fail. * Certain mount options are dependent on the kernel settings of the host running the kubelet and can cause the mount call to fail.

Consume the volume

The persistent volume can be consumed by creating a Pod using the PersistentVolumeClaim created in the previous section. The following manifest creates a Pod and mounts the persistent volume under "/data" within the application container.

2350 Mission College Blvd, Suite 500
Santa Clara, CA 95054
www.hedvig.io

```
kind: Pod
apiVersion: v1
metadata:
  name: centos-test
spec:
  volumes:
  - name: pv-centos-test
    persistentVolumeClaim:
      claimName: pvc-centos-test

containers:
  - name: ctr-centos-test
    image: centos
    command: ["/bin/sh"]
    args: ["-c", "while true; do sleep 10; done"]
    volumeMounts:
    - mountPath: "/data"
      name: pv-centos-test
```


2350 Mission College Blvd, Suite 500
Santa Clara, CA 95054
www.hedvig.io