Running stateful containers with Kubernetes and Hedvig

Solution Whitepaper

Table of Contents

Executive overview 3
Introduction 4
Solution Components 5
Hedvig 5
Hedvig Storage Service 6
Hedvig Storage Proxy 6
Hedvig Virtual Disk 7
Kubernetes 7
Kubernetes Master 9
Kubernetes Node 10
Hedvig deployment options 11
Hyperscale 11
Hyperconverged 11
Kubernetes in Production 12
Health Check and Resource Monitoring 13
Logging 13
Persistent Volumes 14
Networking 14
Service Discovery 14
Load Balancing 14
Creating stateful containers using Hedvig 16
Kubernetes Persistent Volume Framework 16
Persistent Volumes on Hedvig 18
Summary and Conclusion 21
Additional resources 22

Executive overview

Containers are at the center of cloud-ready and cloud-native applications. Container deployments are slowly evolving from small DevOps environments managed by developers to mission-critical, production-ready environments. Kubernetes has proven itself as one of the prominent container orchestrators for managing containerized workloads and services. It orchestrates computing, networking, and storage infrastructure on behalf of user workloads. The next step in container revolution is to find the scalable and persistent storage solution needed to build the stateful containers.

Hedvig provides a container-native persistent storage solution that brings all the enterprise storage capabilities designed to run virtualization and bare-metal workloads at a large scale. Hedvig's instant volume provisioning, cloud-easy storage management, and workflow automation using REST-based APIs enable a storage infrastructure that can be used to run a large number of stateful containers in a cost-effective manner.

Introduction

Hedvig is a software-defined storage platform built on a true, hyperscale architecture that uses modern, distributed systems techniques to meet all primary, secondary, and cloud data needs. The Hedvig Distributed Storage Platform transforms industry-standard x86 or ARM servers into a storage cluster that scales from a few nodes to thousands of nodes.

Hedvig's patented *Universal Data Plane™* architecture stores, protects, and replicates data across any number of private and public cloud data centers. The advanced software stack of the Hedvig Distributed Storage Platform simplifies all aspects of storage with a full set of enterprise data capabilities, which can be granularly provisioned at the application level and automated via a complete set of APIs.

This whitepaper describes the Hedvig Distributed Storage Platform architecture, its enterprise storage capabilities, and how Hedvig provides scale-out storage for containerized applications over Kubernetes.

Solution Components

- *Hedvig*
- *Kubernetes*

Hedvig

The Hedvig Distributed Storage Platform is a modern storage solution for any enterprise compute environment running at any scale. Designed with distributed systems DNA, the Hedvig platform gets better and smarter as it scales, transforming a cluster of x86 or ARM servers into a highly flexible, cost-effective storage system. It is a software-defined storage solution that enables:

- *A scale-out software architecture.*

  Achieve the elasticity needed to grow data services in lock step with changing business requirements.

- *Native, multi-site replication.*

  Natively replicate data among sites to ensure locality and availability.

- *Full automation and orchestration.*

  Automate provisioning and management via orchestration frameworks and APIs for composable infrastructure.

- *Application-specific data services.*

  Match application needs with individual storage policies to meet unique data requirements.

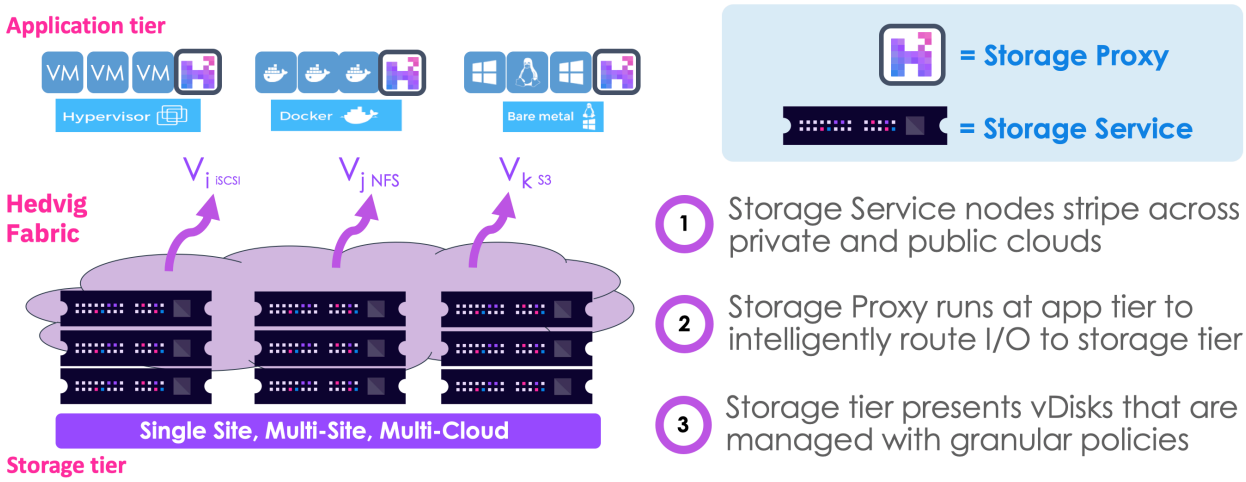

Figure 1: The Hedvig Distributed Storage Platform

The Hedvig Distributed Storage Platform is based on three core components:

- *Hedvig Storage Service*
- *Hedvig Storage Proxy*
- *Hedvig Virtual Disk*

Hedvig Storage Service

The Hedvig Storage Service is a patented distributed systems engine that:

- Scales storage performance and capacity with off-the-shelf x86 and ARM servers.
- Delivers all of the storage options and capabilities required for an enterprise deployment.
- Runs on every storage node.

Hedvig Storage Proxy

The Hedvig Storage Proxy is a lightweight VM or container that:

- Enables access to the Hedvig Storage Service via industry-standard protocols (NFS, iSCSI, S3).

- Enables client-side caching and deduplication with local SSD and PCIe flash resources for fast local reads and efficient data transfers.
- Provides an encryption engine for in-flight and data at rest.

Hedvig Virtual Disk

The Hedvig Virtual Disk is the fundamental abstraction unit of the Hedvig Distributed Storage Platform. Using this feature, you can:

- Spin up any number of concurrent Virtual Disks.
- Have full confidence that each Virtual Disk is thinly provisioned and instantly available.

Kubernetes

Kubernetes is Google's open source portable, extensible platform for managing containerized workloads and services, which facilitates both declarative configuration and automation.

It orchestrates computing, networking, and storage infrastructure on behalf of user workloads. This provides much of the simplicity of Platform as a Service (PaaS), with the flexibility of Infrastructure as a Service (IaaS).

The following figure provides a quick overview of a typical Kubernetes cluster, outlining the different components and how they interact with each other.

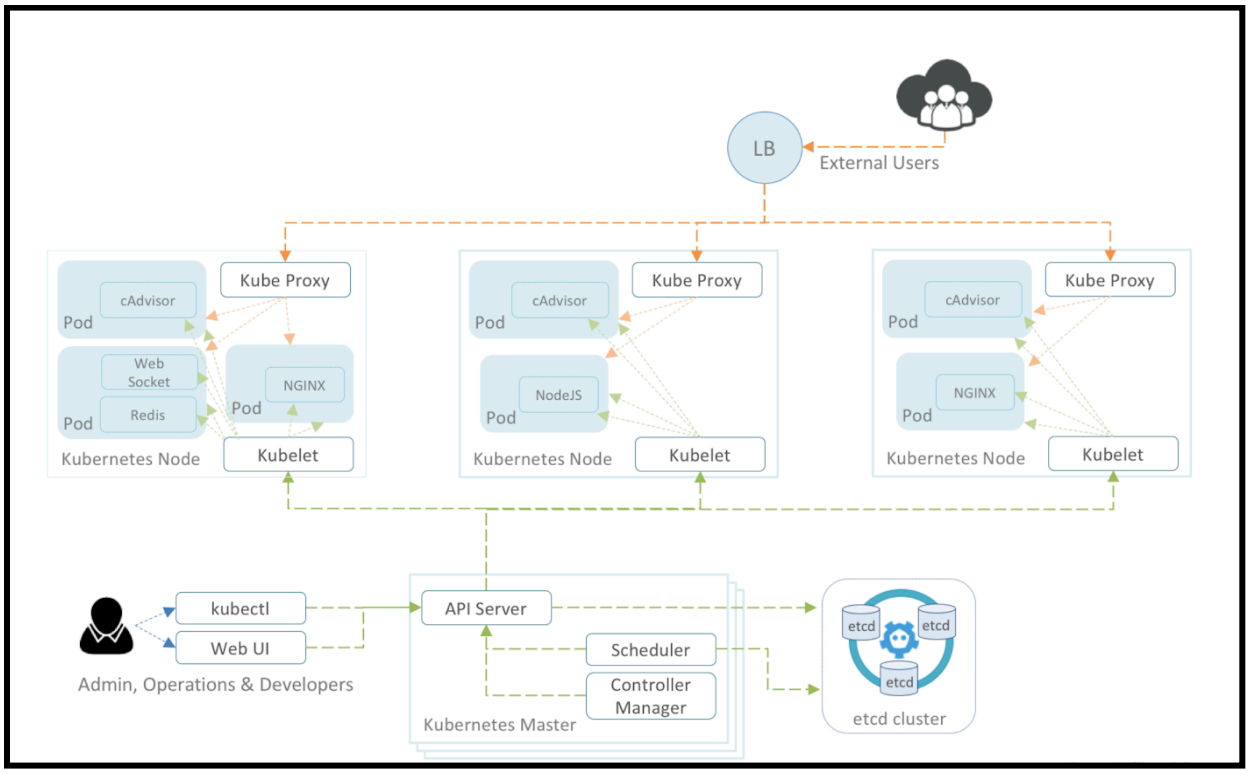

Figure 2: Kubernetes cluster overview

The Kubernetes cluster has two main components:

- *Kubernetes Master*
- *Kubernetes Node*

Kubernetes Master

The Kubernetes master server acts as the primary control plane for the Kubernetes cluster. It serves as the main contact point for administrators and users and also provides many cluster-wide systems for the relatively unsophisticated worker nodes.

Overall, the components on the master server work together to accept user requests, determine the best ways to schedule workload containers, authenticate clients and nodes, adjust cluster-wide networking, and manage scaling and health-checking responsibilities.

Here are the key components of the Kubernetes Master server. These components can be installed on a single server or distributed across multiple servers.

- *Etcd*

  Etcd is a highly available key value store used by Kubernetes to store configuration data that can be accessed by each of the nodes in the cluster. This can be used for service discovery and can help components configure or reconfigure themselves according to up-to-date information. It also helps maintain the cluster state with features such as leader election and distributed locking.

- *API Server*

  The API Server is the Kubernetes REST API entry point that processes operations on Kubernetes objects (Pods, Deployments, Stateful Sets, Persistent Volume Claims, etc.). It is also responsible for making sure that the etcd store and the service details of deployed containers are in agreement. It acts as the bridge between various components to maintain cluster health and disseminate information and commands.

- *Controller Manager*

  The Controller Manager runs control loops that manage objects from kube-apiserver and perform actions to make sure these objects maintain the states described by their specs.

- *Scheduler*

  The process that actually assigns workloads to specific nodes in the cluster is the *Scheduler*. This service reads the operating requirements of the workload, analyzes the current infrastructure environment, and places the work on an acceptable node or nodes.

Kubernetes Node

The Kubernetes Node servers are responsible for running containers. Node servers have a few requirements that are necessary for communicating with master components, configuring container networking, and running the actual workloads assigned to them.

Here are the key components of the Kubernetes node servers.

- *Container Runtime*

  The Container Runtime is responsible for starting and managing containers, applications encapsulated in a relatively isolated, but lightweight operating environment. Each unit of work on the cluster is, at its basic level, implemented as one or more containers that must be deployed. The Container Runtime on each node is the component that finally runs the containers defined in the workloads submitted to the cluster.

- *Kubelet*

  The Kubelet service communicates with the master components to authenticate to the cluster and receive commands and work. Work is received in the form of a manifest that defines the workload and the operating parameters. The kubelet process then assumes responsibility for maintaining the state of the work on the node server. It controls the Container Runtime to launch or destroy containers as needed.

- *Kube Proxy*

  To manage individual host subnetting and make services available to other components, a small proxy service called *kube-proxy* is ran on each node server. This process forwards requests to the correct containers, does primitive load balancing, and is generally responsible for making sure the networking environment is predictable and accessible, but isolated where appropriate.

Hedvig deployment options

Hedvig Distributed Storage Platform components can be configured to support two types of deployments:

- ***Error! Reference source not found.***
- ***Error! Reference source not found.Error! Reference source not found.***

Hedvig also provides the flexibility to leverage both in the same storage cluster.

Hyperscale

Hyperscale deployments scale storage resources independently from application compute resources. With hyperscale, storage capacity and performance scale out horizontally by adding commodity servers running the Hedvig Storage Service.

Application hosts consuming storage resources scale separately with the Hedvig Storage Proxy, allowing for the most efficient usage of storage and compute resources.

Hyperconverged

Hyperconverged deployments scale compute and storage in lockstep, with workloads and applications residing on the same physical nodes as data.

In this configuration, the Hedvig Storage Proxy and the Hedvig Storage Service software are packaged and deployed as VMs on a compute host with a hypervisor installed.

Kubernetes in Production

Kubernetes has established itself as a production-grade container orchestration solution that automates management of cloud-native applications. Kubernetes provides a complete set of features that greatly simplifies automated rollout and rollback, auto-scaling, and self-healing of containerized applications. These features enable application developers to consume the platform at scale through a self-service portal, thereby enhancing developer productivity.

In a typical Kubernetes model, developers have more responsibility for supporting their own applications. Therefore, the Ops team may consider a separation of concerns to clarify roles and to help people focus on their service. To address this, the Ops team may assign a namespace for a new project and allocate resources to that namespace for a particular application service. However, most modern-day applications include multiple containerized services. Therefore, the Ops team may find it more logical to deploy and hand over a separate Kubernetes cluster for multiple related teams and services. This can impact manageability especially when end-to-end microservices applications span multiple clusters.

As more developers are onboarded, it may be hard to identify the appropriate span of control for a specific developer, team, or project. As the number of applications and namespaces grow, lifecycle management of Kubernetes becomes a challenge. Therefore, enterprise readiness for Kubernetes includes a collection of features or attributes for everything from application and resource monitoring, log collection, identity and authorization, to network isolation. These are the features any enterprise would need to successfully run production Kubernetes environments at scale.

The following sections outline the main features to consider when running your own hosted Kubernetes for production.

- *Health Check and Resource Monitoring*
- *Logging*
- *Persistent Volumes*
- *Networking*
- *Service Discovery*
- *Load Balancing*

Health Check and Resource Monitoring

The internal health checking feature in Kubernetes will periodically check your container and restart, if a health check fails. If restarting does not help, Kubernetes will try to restart again while increasing the wait time between restarts. This feature lets you see the reason for the failed health check, which helps with debugging. When a whole node fails in the cluster, Kubernetes will redistribute all containers across the remaining nodes.

The default resources monitoring add-ons in Kubernetes include Heapster, InfluxDB, and Grafana. These add-ons are limited in terms of providing support for queries over stats collected and providing detailed insights. Prometheus is a powerful open-source system for collecting metrics and storing them in a searchable database. With a highly dimensional data model, you can run queries to slice and dice a collected series of data to generate ad-hoc graphs, tables, and alerts.

Logging

The default logging add-ons in Kubernetes include Fluentd, Elasticsearch, and Kibana. Depending on your underlying infrastructure, it is necessary to configure Fluentd to look for interesting log files and parse them. By default, all logs from containers stdout are ingested, and all logs are also labeled with

Kubernetes metadata, which makes them easy to filter through. Elasticsearch can be used to parse application logs and run detailed queries by configuring the applications to log in JSON format.

Persistent Volumes

Kubernetes offers a wide variety of options to set up persistent storage. As a best practice, applications should not use local storage and instead leverage a distributed data store outside the cluster. A detailed description of how Hedvig solves this problem is presented later in this document.

Networking

Kubernetes has a requirement that pods can connect via a "flat networking space." Kubernetes provides a number of ways to implement a network model. Flannel is a very simple overlay network that satisfies Kubernetes requirements. Although it works well in isolated Kubernetes environments, a major drawback of Flannel is that a Pod IP cannot be accessed from outside the Kubernetes cluster.

Project Calico provides a highly scalable networking and network policy solution for connecting Kubernetes pods based on the same IP networking principles as the internet. In Project Calico, broadcast production IP by BGP working on BIRD containers is launched on each Kubernetes node. By default, broadcast is within the cluster only. By setting peering routers outside the cluster, it is possible to access a Pod IP from outside the Kubernetes cluster.

Another solution to enable Pod IP access outside the Kubernetes cluster is Romana. It avoids overlays as well as BGP because it aggregates routes. It uses its own IP address management (IPAM) to maintain the route hierarchy.

Service Discovery

Service Discovery is enabled by Kubernetes using the SkyDNS add-on. This is provided as a cluster internal service, and it is accessible in clusters, such as ClusterIP. By broadcasting ClusterIP by BGP, name resolution also works from outside the Kubernetes cluster.

Load Balancing

Kubernetes introduces a virtual network layer inside the cluster called *services*. Services, together with the DNS add-on, create a powerful way to load balance internal traffic between containers within the

cluster. Containers can reference other containers with a service name without the need to change this reference when you scale the referenced service up and down.

For load balancing traffic from outside the cluster, Kubernetes can automatically provision a load balancer for each of your services in your cloud environment. Creating a load balancer for every service in your cluster might become costly and hard to keep track of. There are multiple choices of external service load balancers for Kubernetes, such as NodePort, LoadBalancer, and Ingress.

Creating stateful containers using Hedvig

On-disk files in a Container are ephemeral, which presents some problems for non-trivial applications when running in Containers. When a Container crashes, kubelet will restart it, but the files will be lost — the Container starts with a clean state. The Kubernetes Persistent Volume framework solves this problem.

The following sections outline the use of Persistent Volumes in Kubernetes and Hedvig:

- *Kubernetes Persistent Volume Framework*
- *Persistent Volumes on Hedvig*

Kubernetes Persistent Volume Framework

The Kubernetes Persistent Volume Framework consists of three main entities:

- *PersistentVolume* — for provisioning volumes on storage systems
- *PersistentVolumeClaim* — for attaching PersistentVolume to application containers
- *StorageClass* — for defining the properties of PersistentVolume

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 Volume and mounts that Volume for the Pod.

A *StorageClass* provides a way for administrators to describe the "classes" of storage that they offer. Different classes might map to quality-of-service levels, or to backup policies, or to arbitrary policies determined by cluster administrators. Each StorageClass contains the fields provisioner, parameters, and reclaimPolicy, which are used when a PersistentVolume belonging to the class must be dynamically provisioned.

Kubernetes' 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 object.

To request storage for Pods, and in turn containers, Persistent Volume Claim (PVC) can either statically or dynamically consume a PV resource based on storage size and access mode. StorageClasses use provisioners that are specific to the storage platform provider to give Kubernetes access to the storage volume being used. For dynamic volume provisioning, PVC informs Kubernetes which storage class to talk to, in order to create PVs using a specific backend storage.

Persistent Volumes on Hedvig

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

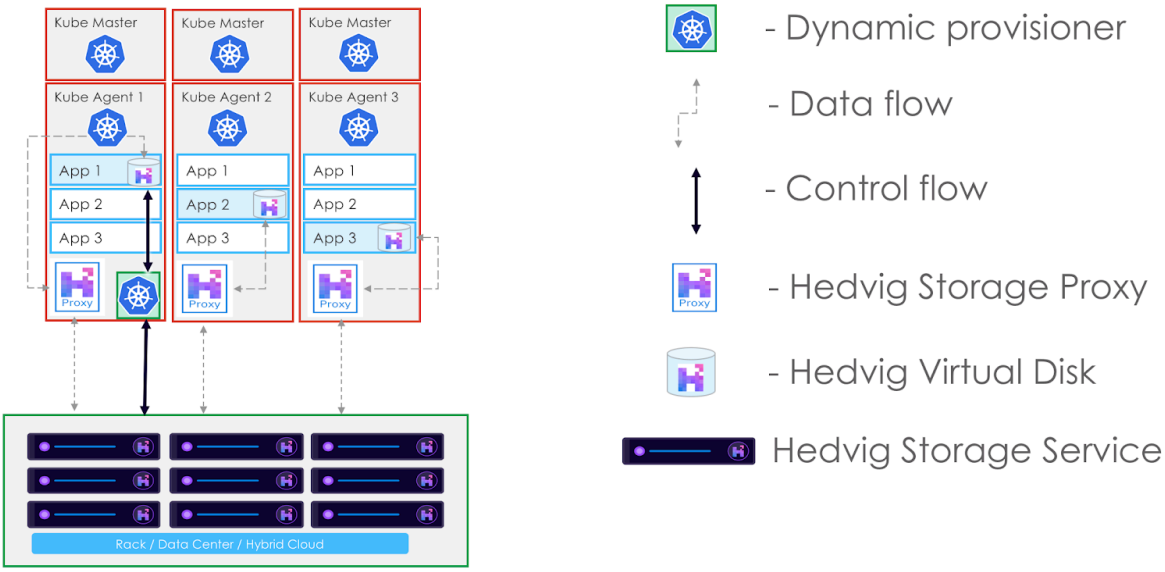

Figure 3: Hedvig Integrated with a Kubernetes cluster

The Hedvig Dynamic Provisioner integrates with Kubernetes and provides a solution to create PVs dynamically with Hedvig as an external storage. The Dynamic Provisioner is installed as a Pod in the Kubernetes cluster. After installation, you can use it by mentioning it in a storage class template, which will then be consumed by PVC. When a Pod requests a storage volume using the previously mentioned PVC, a PV is created dynamically meeting those requirements, and it will be provided to the Pod in real time.

In addition to the Dynamic Provisioner, the Hedvig Storage Proxy is deployed as a Daemonset in the Kubernetes cluster. This ensures that Kubernetes spawns one storage proxy Pod on every Kubernetes node. When a PV is dynamically created and mounted in the application Pod, any I/O operations to that PV are handled by the storage proxy Pod running on the same Kubernetes node as the Application Pod. If the Application Pod is rescheduled or restarted on a new Kubernetes node, it is reattached to its PV through the storage proxy Pod running on the new Kubernetes node.

Storage volume attributes can be tuned using storage classes. Here is an example of a storage class with deduplication and compression enabled for the dynamically created storage volumes:

```
apiVersion: storage.k8s.io/v1beta1
kind: StorageClass
metadata:
  name: sc-hedvig-compress-dedup
provisioner: hedvig.io/provisioner
parameters:
  backendType: "hedvig-block"
  compressed: "true"
  dedupEnable: "true"
```

Here is an example of a PersistentVolumeClaim that uses the Hedvig Dynamic Provisioner to dynamically create a PersistentVolume using the StorageClass defined above:

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

Here is a list of the Hedvig volume attributes that can be configured using storage classes.

key	values	default value	notes
dedupEnable	true/false	false	
compressed	true/false	false	
cacheEnable	true/false	false	
rf (replication factor)	1 to 6	3	
rp (replication policy)	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	""	

Summary and Conclusion

Hedvig brings enterprise-hardened storage features to containerized workloads, and Kubernetes simplifies deployment and management of containers at a large scale. Together, Hedvig and Kubernetes empower enterprises to build a production-ready container infrastructure for stateful applications that is designed to be cloud-easy and web-scale.

Additional resources

Hedvig's architecture overview:

https://www.hedvig.io/technical-overview-whitepaper

Hedvig's Docker solution brief:

https://www.hedvig.io/hubfs/Website_Resources/Updated Assets 2018/Hedvig - Solutions Brief - Docker.pdf