White Paper

Version 0.7

Container Attached Storage (CAS) and OpenEBS

OpenEBS

Introduction

The following paper discusses the fundamental architecture and approach of OpenEBS - a solution intended to simplify and otherwise improve the management of data for stateful workloads in container environments.

Originally containers and Kubernetes were optimized for stateless applications and restricted enterprises to use container technology for workloads such as websites and user interfaces. More recently stateful workloads have been increasingly run on Kubernetes for a variety of reasons:

- Improved ease of use. Kubernetes is being broadly adopted in large part because it is proven to simplify the operation of software in a distributed manner. These same benefits can apply to the operation of distributed stateful systems.
- Kubernetes enhancements. Since the introduction of "pet sets", now known as stateful sets, in Kubernetes 1.3, there have been countless improvements to Kubernetes for stateful workloads.
- Emergence of OpenEBS and similar projects. Most if not all storage vendors have started to ship connector to Kubernetes to enable at least basic use cases for their storage systems and projects such as OpenEBS have emerged to more fully leverage the possibilities of Kubernetes as a platform for stateful workloads.

However challenges remain. Typical storage systems themselves are complex distributed systems that require special skills to install, run and operate. Kubernetes is a distributed system meant to standardize and simplify the running of distributed systems. The essence of the Container Attached Storage (CAS) architecture is to use Kubernetes itself to deliver storage capabilities to stateful workloads running on Kubernetes. The benefits include reduced total cost of ownership and, crucially, improved agility thanks to removing impediments to the dynamic operations of Kubernetes based systems.

While a CAS solution such as OpenEBS runs on Kubernetes it is worth noting that some users choose to segregate OpenEBS storage onto its own segments of their Kubernetes deployment. In other words, a hyper converged approach is typical however is not required.

Comparison of CAS to Direct Attached Storage and Scale out

Container Attached Storage (CAS) is software that includes containerized storage targets that can themselves be disaggregated and run as multiple services and multiple microservice-based storage replicas that store the data and that, again, can be independently scaled and scheduled. Developers and operators orchestrate CAS using Kubernetes like any other workload or container which helps these groups to increase their autonomy and control and hence boosts their agility. Often each team will own a workload - and as a part of that they will own an OpenEBS deployment as well.

In many regards CAS provides the benefits of DAS or Direct Attached Storage such as keeping data close to the workload and reducing the performance tax, operational risk and costs of scale out storage systems. Some historical perspective may be useful; scale out storage systems were built at a time in which applications were not distributed or able to scale out and disk drives were the performance bottleneck of many architectures. By comparison, today applications running in Kubernetes are by definition distributed systems and the up to 150 IOPS of a single disk drive has been replaced by often hundreds of thousands of IOPS offered by solid state media. Additionally the size of a database in a scale up architecture application is typically much larger than the size of the sometimes hundreds of databases that often work together in today's scale out applications. In short, scale-out storage architectures were designed to address needs that typically no longer exist in a Kubernetes based environment. The CTO of MayaData / OpenEBS and others have spoken on these subjects in a variety of technical settings.

"Scale-out storage architectures were designed to address needs that typically no longer exist in a Kubernetes based environment."

Additionally, in many non CAS storage solutions the storage software requires that Kubernetes Persistent Volumes are tightly coupled to the Kernel modules of the Kubernetes node. This means that the storage is essentially wired into the environment and cannot be safely rescheduled by Kubernetes; indeed most such solutions require the use of a seperate scheduler as opposed to Kubernetes and a separate place to store the state of an environment as opposed to the Kubernetes native use of etcd.

It is as if these scale out storage architectures require users to run a special purpose idiosyncratic version of Kubernetes - one that requires special knowledge to operate and that constrains the operations of Kubernetes itself - in order to provide services to Kubernetes.

"It is as if these scale out storage architectures require users to run a special purpose idiosyncratic version of Kubernetes... in order to provide services to Kubernetes."

CAS Benefits

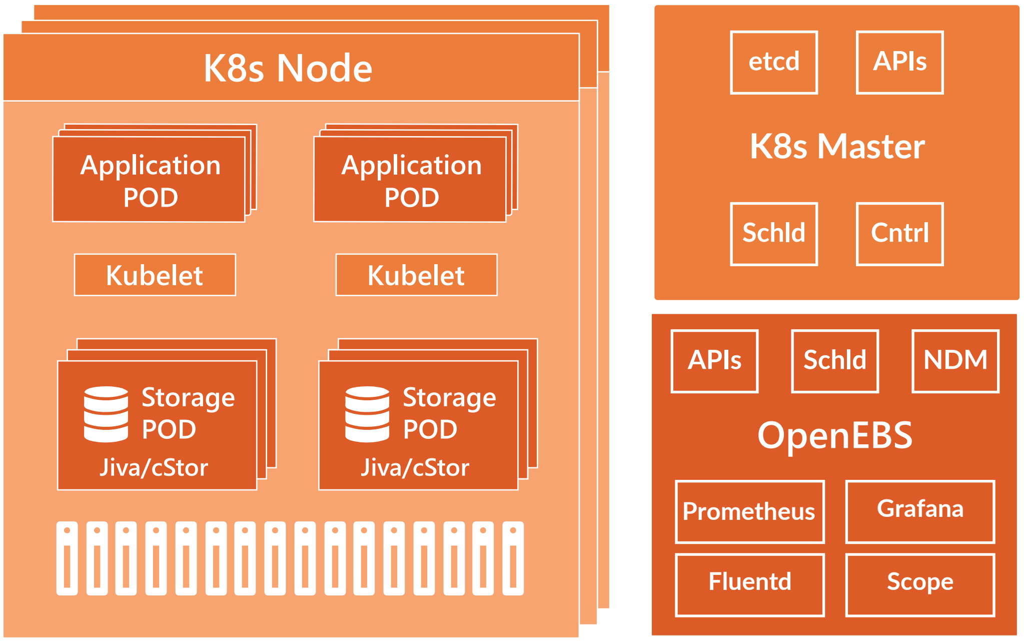
bit.ly/cncf-cas-blog

OpenEBS is a storage solution built for stateful workloads on containers that utilizes containers and Kubernetes. There are many benefits which users of OpenEBS Container Attached Storage are reporting including:

- Immediate provisioning. As quick as less than one minute thanks in part to integration with Helm Charts.
- Per workload and per team control. Each workload and team has its own OpenEBS with their own storage policies. This approach is consistent with DevOps governance and culture.
- Other benefits of a containerized and Kubernetes orchestrated architecture. These include simpler upgrades, higher velocity of development, independent scaling, and cloud independence.
- Other enterprise class storage features. OpenEBS also can include capabilities such as snapshots and cloning and replication for data recovery and high availability such as cross availability zone operations and cross cloud replication and migration.

OpenEBS Architecture

OpenEBS is the most popular open source Container Attached Storage or CAS. As a part of this approach, each volume has a dedicated target controller POD and a set of replicas. OpenEBS is simple to operate and to use largely because it looks and feels like other cloud native and Kubernetes friendly projects.



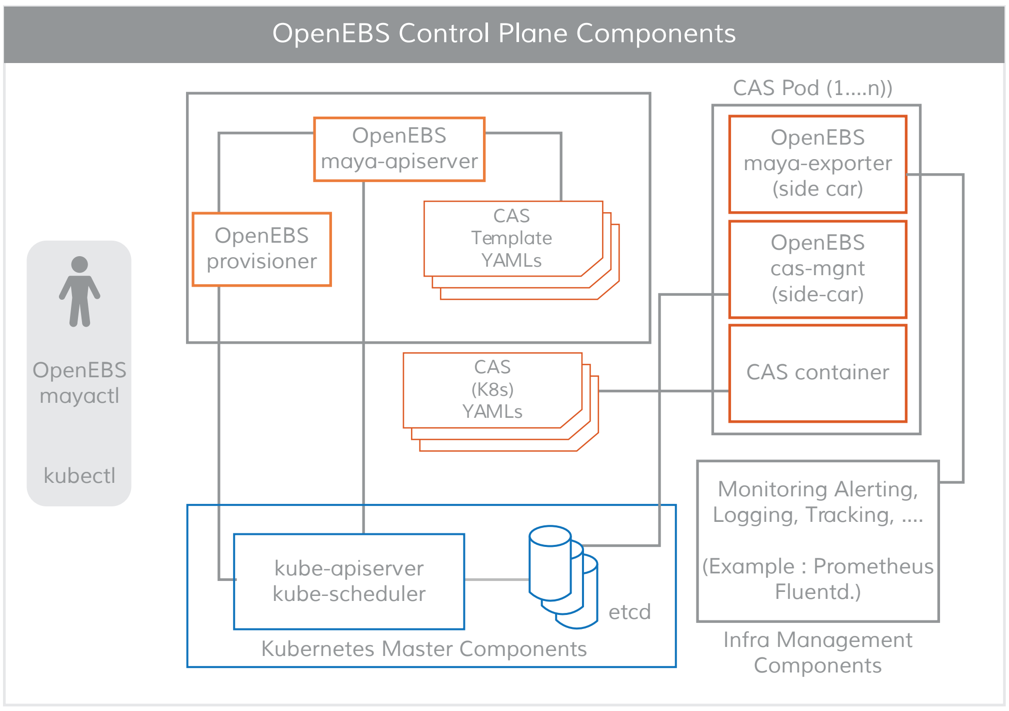
OpenEBS Architecture

OpenEBS has many components, which can be grouped into the following categories:

- Control plane components. Provisioner, API Server, and volume sidecars
- Data plane components. Jiva and cStor - also known as pluggable storage engines
- Node disk manager. Discover, monitor, and manage the media attached to the Kubernetes node; this capability is used by OpenEBS and is being contributed upstream into Kubernetes itself
- Integrations with cloud native tools. Current integrations include Prometheus, Grafana, Fluentd, and WeaveScope

OpenEBS control plane

The control plane of an OpenEBS cluster is often referred to as Maya. Maya is responsible for provisioning volumes, associated volume actions such as taking snapshots, making clones, creating storage policies, enforcing storage policies, exporting the volume metrics for consumption by prometheus/grafana, and so on.


OpenEBS Control Plane Components
OpenEBS maya-apiserver
OpenEBS provisioner
CAS Template YAMLs
CAS (K8s) YAMLs
CAS Pod (1....n))
OpenEBS maya-exporter (side car)
OpenEBS cas-mgnt (side-car)
CAS container
OpenEBS mayactl
kubectl
kube-apiserver kube-scheduler
etcd
Kubernetes Master Components
Monitoring Alerting, Logging, Tracking, ...
(Example : Prometheus Fluentd.)
Infra Management Components

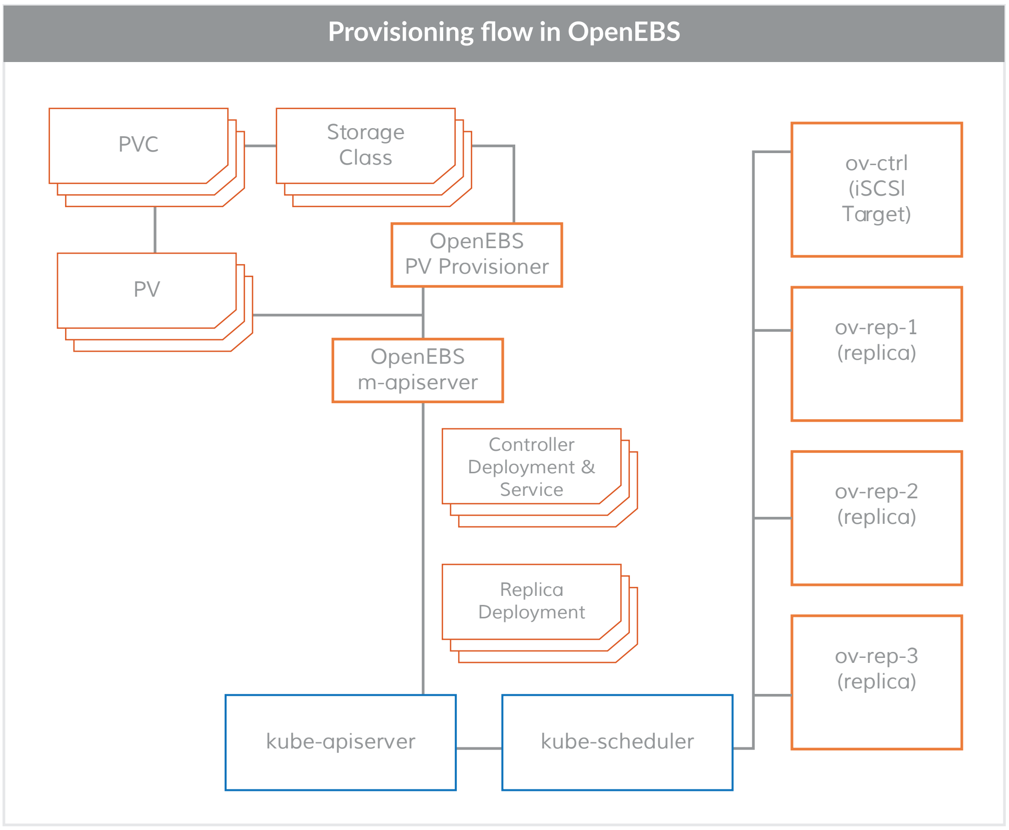
OpenEBS provides a dynamic provisioner, which integrates into Kubernetes just like any other external storage plugin. The primary task of an OpenEBS PV provisioner is to initiate volume provisioning to application PODS and to implement the Kubernetes specification for PVs. The m-apiserver exposes storage REST API and handles the bulk of volume policy processing and management.

The connectivity between the control plane and the data plane uses a Kubernetes sidecar pattern. There are a couple of scenarios as follows in which the control plane needs to communicate with the data plane.

- For volume statistics such as IOPS, throughput, latency etc - achieved through volume-exporter sidecar
- For volume policy enforcement with volume controller pod and disk/pool management with the volume replica pod - achieved through volume-management sidecar(s)

OpenEBS PV Provisioner:

The OpenEBS PV Provisioner runs as a POD and makes provisioning decisions. The developer constructs a claim with the required volume parameters, chooses the appropriate storage class via their declarative yaml specification. The OpenEBS PV dynamic provisioner interacts with the maya-apiserver to create deployment specifications for the volume target controller pod and volume replica pod(s) on appropriate nodes. Scheduling of the volume pods (target controller/replica) can be controlled using annotations in PVC specification, details of which are discussed in a separate section. Currently the OpenEBS provisioner supports only one type of binding i.e. iSCSI however it is extensible.

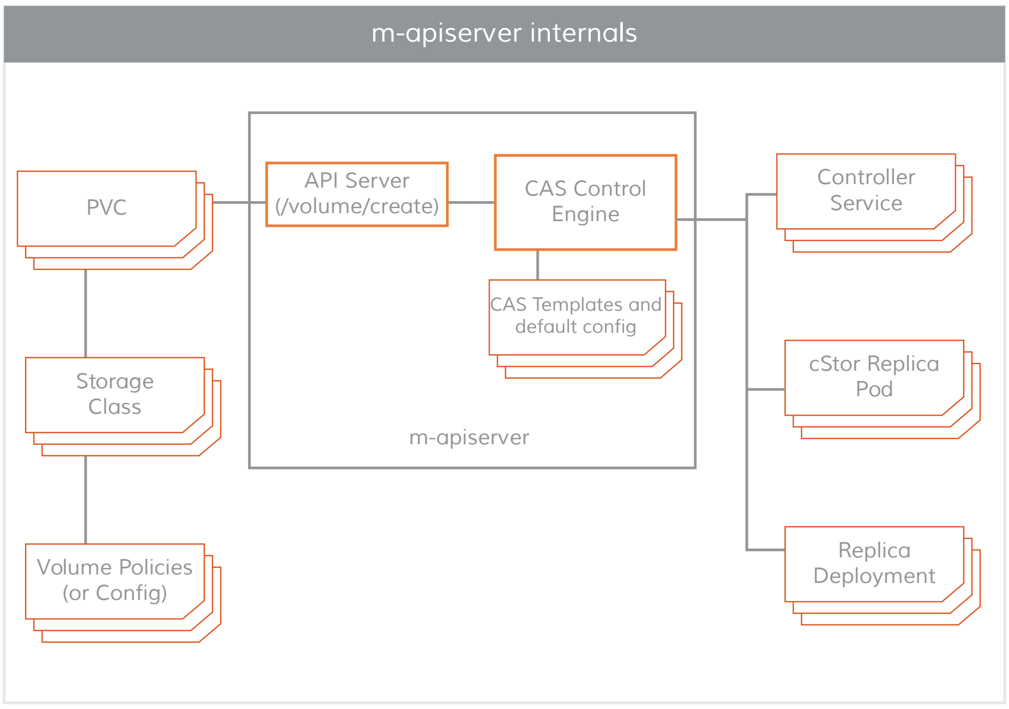


Maya-API Server:

The Maya or m-apiserver runs as a POD. As the name suggests, m-apiserver exposes the OpenEBS REST APIs. m-apiserver is also responsible for creating deployment specification files required for creating the volume pods. After generating these specification files, it invokes kube-apiserver for scheduling the pods accordingly.

At the completion of volume provisioning by the OpenEBS PV provisioner, a Kubernetes object PV is created and is mounted on the application pod. The PV is hosted by the target controller pod which is supported by a set of replica pods in different nodes. The target controller pod and replica pods are part of the data plane and are described in more detail in the Storage Engines section.

Another important task of the m-apiserver is volume policy management. OpenEBS provides very granular specification for expressing policies. m-apiserver interprets these yaml specifications, converts them into enforceable components and enforces them through volume-management sidecars or using the Kubernetes constructs like resource requests and limits.

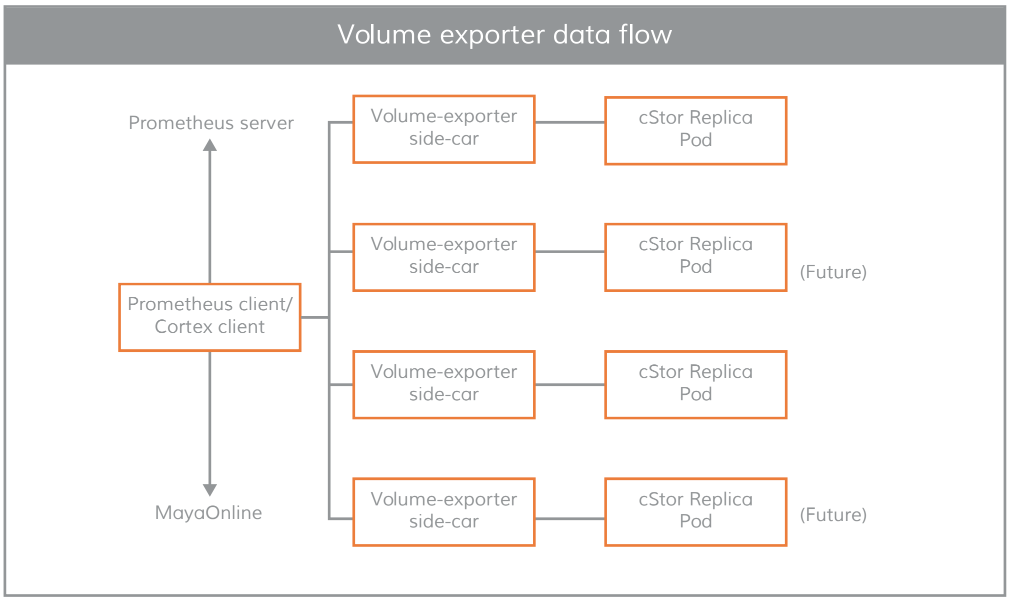


Maya Volume Exporter:

The Maya Volume Exporter runs as a sidecar for each of the storage target controller pods (cStor/Jiva). These sidecars enable the REST endpoints for fetching statistics by connecting to the data engines. The granularity of statistics is at the volume level. Some example statistics are:

- volume read latency
- read IOPS
- read block size
- capacity statistics
- volume write latency
- write IOPS
- write block size

These statistics are typically pulled either by the Prometheus that is installed and configured during OpenEBS installation or by the Weave Cortex agent that is installed and configured during connectivity to MayaOnline (a free solution for OpenEBS users for visualization and management).



Volume Management Sidecar:

Sidecars are also used for passing target controller configuration parameters and volume policies to the volume controller pod which is a data plane and for passing replica configuration parameters and replica data protection parameters to the vol- ume replica pod.

OpenEBS data plane

The OpenEBS data plane is responsible for the volume IO path. A storage engine implements the actual IO path in the data plane. Because OpenEBS is itself containerized and delivers functionality on a per workload and per team basis, an architecture that supports a pluggable storage engine was developed. As of the summer of 2018, OpenEBS provides two storage engines that can be plugged in easily. These are called Jiva and cStor. Certain existing storage vendors may choose to develop or "port" their storage engine functionality for inclusion in OpenEBS environments as well. The Jiva and cStor storage engines run completely in Linux user space and are based on microservices. As mentioned above, this user space requirement means that underlying cloud services or Kubernetes distributions do not need kernel level adjustments to be able to run OpenEBS. The kernel is not "tainted."

Jiva:

The Jiva storage engine is developed with Rancher's LongHorn and gotgt as the base. The entire Jiva engine is written in GO language and runs in the user space. LongHorn controller synchronously replicates the incoming IO to the LongHorn replicas. The replica considers a Linux sparse file as the foundation for building the storage features such as thin provisioning, snapshotting, rebuilding etc.

cStor:

cStor is a storage engine built with proven building blocks of storage components such as a BSD based Multi-threaded iSCSI protocol stack and a DMU layer of user space ZFS. cStor gives provable data integrity, CoW based snapshots, stores its data to raw disks and more. Common use cases include larger environments using snapshots and clones as a part of test, deploy and operate pipelines; for example clones are often used with DBs running on OpenEBS in staging pipelines.

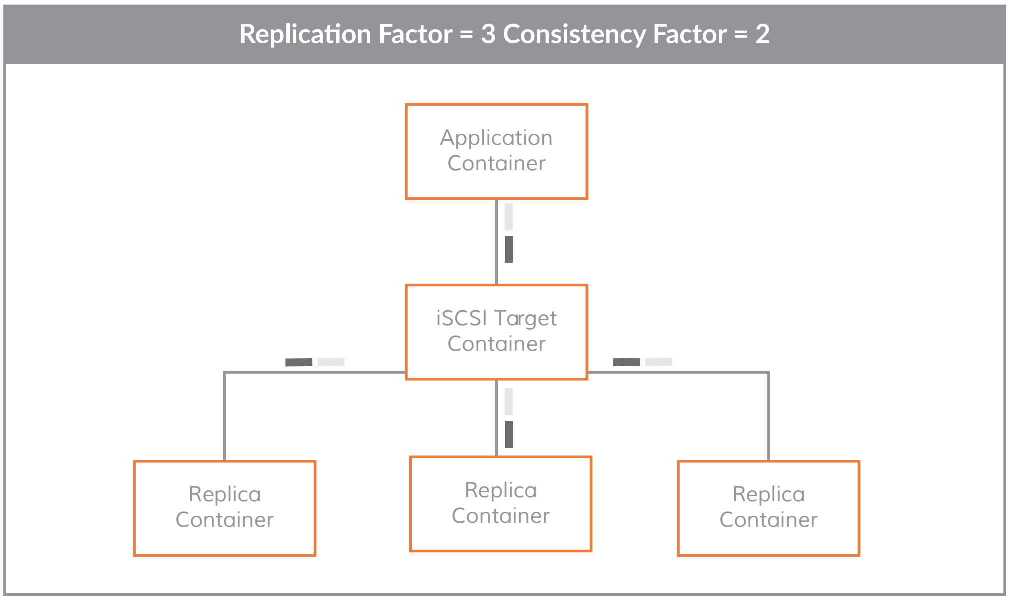
IO path:

The data path includes

- Application running in a container
- iSCSI target container
- Storage container (replica)

These three or more containers are connected via TCP/IP connections. Based on the replication factor that has been established, the iSCSI target connects to those many replicas. Based on the consistency factor, each write IO must be committed to a certain number of replicas equal to consistency factor. In the following diagram the replication factor is 3 and consistency factor is 2. The replica and iSCSI target exchange information such as block size, capacity etc as part of handshake operation. The iSCSI target will not indicate that it is ready to receive IOs until it in turn sees the number of replicas online needed to address the consistency factor.

Once the initial setup is complete, the application starts IOs to the iSCSI target, and the iSCSI target in turn forwards the IO request to online replicas. Replicas read/write data from/to attached disk pool, and send ACK back to iSCSI target. The iSCSI target waits for a number of ACKs equal to consistency factor before it sends ACK to application.



CAS Templates

Using OpenEBS helm charts, one can easily install OpenEBS onto K8S cluster and start provisioning basic storage volumes. However, when the need expands to dynamic storage features that includes provisioning, expansion, purging, snapshot, backups, migration, upgrades and so on, then simple charts will not help. The K8S developers may expect s operators to be provided for each individual feature and use these operators to tweak the feature for their application or file-system. This process is be too tedious and not scalable.

OpenEBS has thought through this problem and provides a solution that bridges the gap between features versus making use of them based on local needs. This solution is also the one that OpenEBS uses as the building blocks of its operators. This building block is called as CAS Templates nicknamed as `cast`. As the nickname suggests, it's the tool of choice to mould any requirement to give it a concrete implementation. CAS Templates are Kubernetes custom resources that can accept tunables, configs, options, etc and at the same time prescribe a workflow to execute various tasks (i.e. commands or APIs) in order. It also has the ability to perform rollback and fallback depending on specific conditions set into these tasks. In addition, OpenEBS makes use of cast to simplify install and upgrades. This also speaks a lot on how individual teams can customise OpenEBS install and upgrades based on their needs.

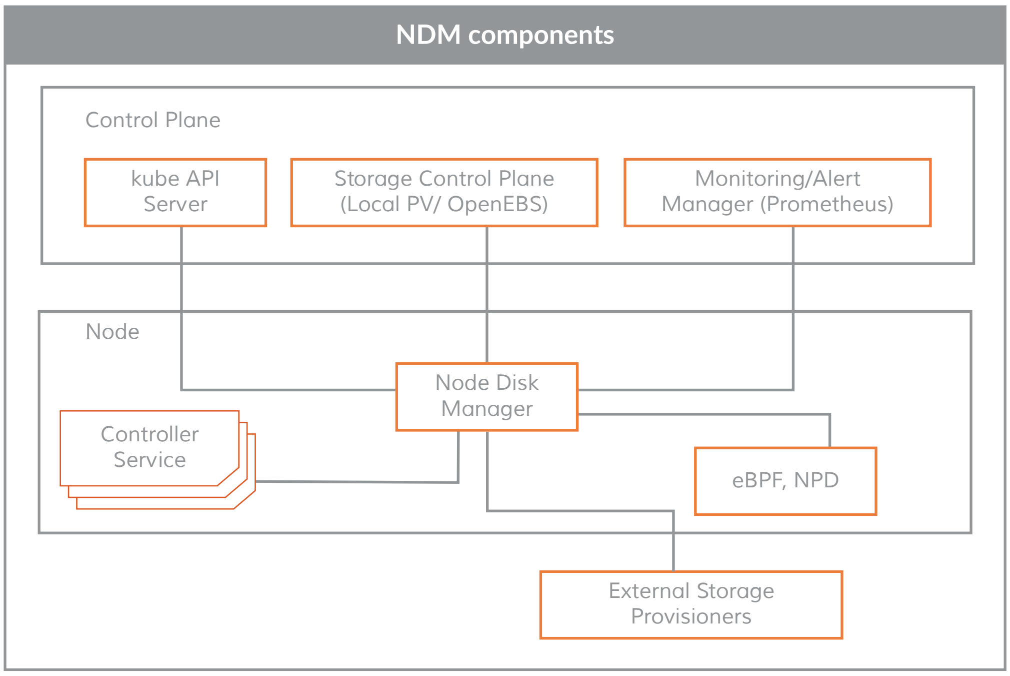
Below is a CAS Template that manages the workflow of deleting a OpenEBS cStor based volume:

```
apiVersion: openebs.io/v1alpha1
kind: CASTemplate
metadata:
  name: cstor-volume-delete-default-0.7.0
spec:
  defaultConfig:
  - name: RunNamespace
    value: {{env "OPENEBS_NAMESPACE"}}
  taskNamespace: {{env "OPENEBS_NAMESPACE"}}
  run:
    tasks:
    - cstor-volume-delete-listcstorvolumecr-default-0.7.0
    - cstor-volume-delete-listtargetservice-default-0.7.0
    - cstor-volume-delete-listtargetdeployment-default-0.7.0
    - cstor-volume-delete-listcstorvolumereplicacr-default-0.7.0
    - cstor-volume-delete-deletetargetservice-default-0.7.0
    - cstor-volume-delete-deletetargetdeployment-default-0.7.0
    - cstor-volume-delete-deletecstorvolumereplicacr-default-0.7.0
    - cstor-volume-delete-deletecstorvolumecr-default-0.7.0
  output: cstor-volume-delete-output-default-0.7.0
```

Node Disk Manager (NDM)

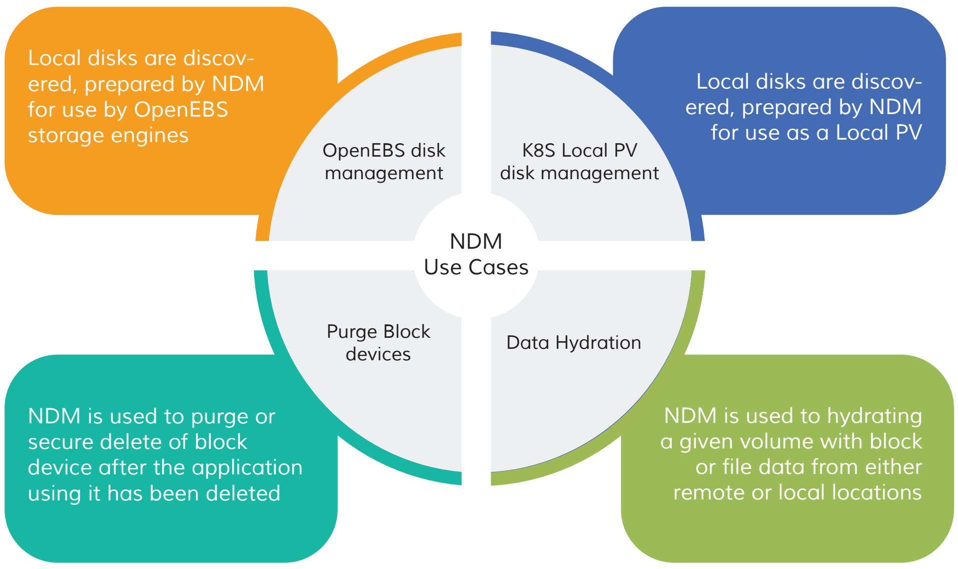
Node Disk Manager or NDM in OpenEBS ad upstream in Kubernetes enables underlying disk or storage media management in a Kubernetes way by using Kubernetes constructs. NDM is being contributed upstream to Kubernetes and extends etcd for example as a central store of disk status or inventory.

NDM runs as a daemonset on Kubernetes nodes which are selected for disks or storage media management. The disks are managed through YAML manifests and are added as Kubernetes custom resources (CRs) into an etcd database. NDM helps bring an platform agnostic disk (or storage device) management functionality to Kubernetes. For example - local SSDs or other media attached to the instances (or Kubernetes node) may differ depending the vendor. NDM encapsulates these differences and provides an standard interface to manage the disks. The disk custom resources can then be used by higher level storage provisioners, be it local PV provisioners or OpenEBS Provisioners (like the cStor operator), without having to worry about vendor details.


NDM components
Control Plane
kube API Server
Storage Control Plane (Local PV/ OpenEBS)
Monitoring/Alert Manager (Prometheus)
Node
Node Disk Manager
Controller Service
eBPF, NPD
External Storage Provisioners

NDM can be used as well to facilitate the monitoring of the health of the disks using tools such as Prometheus or queried by Alert Managers or by other operators to provide telemetry information to aid in troubleshooting.

NDM can be completely operated via Kubernetes without having to learn any special skills/tools. NDM is being enhanced to support automatic provisioning of external disks into the nodes - beyond discovering attached disks. NDM containers are also designed so that they can be used as side cars or init-containers or as Kubernetes jobs to carry out some specific storage management tasks such as purging or secure delete of block devices after the application using it has been deleted or even hydrating a given volume with block or file data from either remote or local locations.


Local disks are discovered, prepared by NDM for use by OpenEBS storage engines
OpenEBS disk management
K8S Local PV disk management
Local disks are discovered, prepared by NDM for use as a Local PV
NDM Use Cases
Purge Block devices
Data Hydration
NDM is used to purge or secure delete of block device after the application using it has been deleted
NDM is used to hydrating a given volume with block or file data from either remote or local locations

OpenEBS Benefits

Agility:

Each storage volume in CAS has a containerized storage target controller and corresponding containerized replicas. Hence, maintenance and tuning of the resources around these components are truly agile. The Kubernetes rolling upgrades capability enables seamless upgrades of storage target controller and storage replicas. Also, common operational tasks such as the tuning of resources such as CPU and memory, are performed in the same way other workloads are tuned - using container cGroups.

Perhaps even more importantly - the architecture of OpenEBS and other CAS solutions fits and supports best practices for organizations embracing a devops culture and set of practices. For example, teams select their own tools and are autonomous to the extent possible in operations.

Cross environment portability:

Storage can easily be moved between different machines and even between different Kubernetes deployments thanks to this architecture. Recent results from an OpenEBS user survey indicate that over 76% of users with more than 4 nodes deployed currently deploy or have deployed OpenEBS on more than one Kubernetes environment. Having a common storage layer that itself can be moved increases flexibility.

Granularity of storage policies:

Containerizing the storage software and dedicating the storage target controller to each volume brings maximum granularity in storage policies. Thanks to the CAS architecture, you can configure all storage policies on a per-volume basis. In addition, you can monitor storage parameters of every volume and dynamically update storage policies to achieve the desired result for each workload. The control of storage throughput, IOPS, and latency is also increased with this additional level of granularity in the volume storage policies.

Avoid Lock-in:

Avoiding cloud vendor lock-in is a commonly stated goal for many users and most enterprises; the rise of Kubernetes has been attributed in part to the desire to use a common broadly adopted set of APIs to enable cloud independence and multi-cloud approaches However, the data of stateful applications remains dependent on the cloud provider and technology;

typically different approaches are used for each cloud service provider and each on premise Kubernetes deployment as well, increasing operations costs, decreasing developer agility, and raising the risk of lock-in due to the substantial engineering that would be required to move workloads off of any particular environment. As mentioned above, OpenEBS provides a common layer so applications perform the more similarly than otherwise would be the case whether the backend is EBS or on premise disk or some other backing store from another cloud provider. Additionally, the OpenEBS storage target controllers can migrate data in the background and even live migration becomes an achievable task that has been demonstrated by OpenEBS and is on the immediate road map for OpenEBS. In other words, stateful workloads can be moved from one Kubernetes cluster to another in a non-disruptive way, dramatically improving the leverage of Kubernetes users when negotiating pricing with Kubernetes service providers.

Cloud native integrations:

Containerizing the storage software and dedicating the storage target controller to each volume brings maximum granularity in storage policies. Thanks to the CAS architecture, you can configure all storage policies on a per-volume basis. In addition, you can monitor storage parameters of every volume and dynamically update storage policies to achieve the desired result for each workload. The control of storage throughput, IOPS, and latency increases with this additional level of granularity in the volume storage policies.

Kubernetes integrated Space efficient Snapshots and clones:

OpenEBS implements the Kubernetes snapshot provisioner specification. Snapshot are often used for data protection as well as for data recovery and, along with clones, can be used to add stateful workloads into existing pipelines such as CI/CD pipelines that implement a blue/green approach to deployments. Since snapshots freeze an in-time copy of data/volume, they may increase storage usage depending upon implementation. However the cStor storage engine is the only CAS available that uses a widely deployed open source based Copy on Write (COW) mechanism for writes and data modifications, which means that snapshots in cStor are created instantaneously without any space tax or additional processing. It can be helpful to think of these snapshots and clones as simply pointers to already existing data. Thanks to this COW mechanism, the active file system and any snapshots both share the same blocks as long as blocks are not getting modified by the active file system. This approach is so efficient that it allows the cStor storage engine to support unlimited snapshots.

"It can be helpful to think of these snapshots and clones as simply pointers to already existing data....This approach is so efficient that it allows the cStor storage engine to support unlimited snapshots."

Superior resilience:

The replication capabilities of OpenEBS are used for high availability such as cross Available Zone (AZ) availability as well as for data recovery and can be extended to simplify cross cloud or Kubernetes migration as well. Replication to each replica in the cStor storage engine is synchronous i.e. data is written to all replicas before asked to client. Unlike most distributed storage architectures where the storage system is off-line as it recovers from a a crash restart, the OpenEBS cStor storage engine does not go off-line while rebuilding. Instead, the storage is available for reads and writes for the workload moments after a crash. The cStor storage engine has a mechanism to heal in the background while serving read and writes to the workload.

"Unlike most distributed storage architectures where the storage system is off-line as it recovers from a crash restart, the OpenEBS cStor storage engine does not go off-line while rebuilding... The cStor storage engine has a mechanism to heal in the background while serving read and writes to the workload."

Conclusion

Containerization and Kubernetes are enabling enterprises to accelerate their agility while controlling their operating costs. Container Attached Storage (CAS) and the leading open source CAS project OpenEBS extends containers and Kubernetes to manage stateful workloads much in the way that stateless workloads are managed. The result can be improved data resilience, reduced cloud and Kubernetes vendor lock-in, and dramatic gains to developer agility thanks to the granular control offered by such approaches.

You can learn more and can join the OpenEBS community at www.openEBS.io. To learn more about the MayaData Data Agility Platform that extends OpenEBS with chaos engineering from the open source Litmus and automated visualization and control from MayaOnline, please visit www.mayadata.io.

MAYADATA

twitter.com/mayadata_inc

medium.com/mayadata

linkedin.com/company/mayadata/

OpenEBS

twitter.com/openebs

blog.openebs.io

slack.openebs.io