Container Attached Storage (CAS) for stateful applications on containers

Uma Mukkara
@umamukkara @openebs
MayaData Inc.
Agenda

- Who
- Container Attached Storage
  - What?
  - Why?
  - How?
- Reference implementation of CAS
- Example of Blue Green Deployment
Who?

- MayaData delivers Data Agility
- Formerly known as CloudByte
- Sponsor of **OpenEBS** and **Litmus** projects
- Large Bangalore based OSS contributor
- [https://mayaonline.io](https://mayaonline.io) is SaaS software used to manage stateful workloads on Kubernetes
Who?

- Popular open source storage project for stateful applications on Kubernetes
- Team of 50+ hackers with storage and DevOps experience

OpenEBS

slack.openebs.io

- Leading project in CAS category
- 110+ contributors
- 5K+ github stars
- 1M+ docker pulls
Who?

- Chaos engineering for stateful applications
- Extends your CI/CD pipelines
Introduction to CAS

- CAS - Container Attached Storage
- It is a new architecture in storage domain
- Applicable only to containers
Genesis of CAS

What if storage for container native applications was itself container native?
Storage trends

Microservices / DevOps / Cloud / Kubernetes / incredible acceleration of storage media have changed everything about shared storage.

Shared storage is not optimal now:

- It used to accelerate your storage
  - now it is slower than DAS
- It used to be how you kept your app resilient
  - now it increases your blast radius
- It used to be run by one of many IT silos
  - now those silos are fading away
DAS vs Distributed

**DAS**

**Benefits:**
- Simple
- Ties application to storage
- Predictable for capacity planning
- App deals with resiliency
- Can be faster

"We have ~100k nodes of Cassandra alone, and use DAS because it is easier even if it burns energy and capEx."

Other example DAS users moving to Kubernetes:

- Netflix
- Booking.com
- Tenable
- Walmart
- Reddit
DAS vs Distributed

DAS

Benefits:
- Simple
- Ties application to storage
- Predictable for capacity planning
- App deals with resiliency
- Can be faster

Concerns:
- Under-utilized hardware
  - 10% or less utilization
- Wastes data center
- Difficult to manage
- Lacks storage features
- Cannot be repurposed - made for one workload
- Does not support mobility of workloads via containers
- Cross cloud impossible

“We have ~100k nodes of Cassandra alone, and use DAS because it is easier even if it burns energy and capEx.”

Other example DAS users moving to Kubernetes:

- Netflix
- Booking.com
- Tenable
- Walmart
- Reddit
## DAS vs Distributed

### DAS

**Benefits:**
- Simple
- Ties application to storage
- Predictable for capacity planning
- App deals with resiliency
- Can be faster

**Concerns:**
- Under-utilized hardware
  - 10% or less utilization
- Wastes data center
- Difficult to manage
- Lacks storage features
- Cannot be repurposed - made for one workload
- Does not support mobility of workloads via containers
- Cross cloud impossible

### Distributed

**Benefits:**
- Centralized management
- Greater density and efficiency
- Storage features such as:
  - Data protection
  - Snapshots for versioning

**Concerns:**
- Additional complexity
- Enormous blast radius
- Expensive
- Requires storage engineering
- Challenged by container dynamism
- No per microservice storage policy
- I/O blender impairs performance
- Locks customers into vendor
- Cross cloud impossible
DAS vs Distributed

**DAS**

**Benefits:**
- Simple
- Ties application to storage
- Predictable for capacity planning
- App deals with resiliency
- Can be faster

**Concerns:**
- Under-utilized hardware
  - 10% or less utilization
- Wastes data center
- Difficult to manage
- Lacks storage features
- Cannot be repurposed - made for one workload
- Does not support mobility of workloads via containers
- Cross cloud impossible

**Distributed**

**Benefits:**
- Centralized management
- Greater density and efficiency
- Storage features such as:
  - Data protection
  - Snapshots for versioning

**Concerns:**
- Additional complexity
- Enormous blast radius
- Expensive
- Requires storage engineering
- Challenged by container dynamism
- No per microservice storage policy
- I/O blender impairs performance
- Locks customers into vendor
- Cross cloud impossible
Huge blast radius

Not tunable for containers

Latency & Big 0

Lock-in & not x-cloud

Monolithic mess
Yet Another Scale Out Storage System
What is needed?

- Like DAS
- No blast radius
- Container friendly

CAS
CAS = Containers & DAS\(^+\) & Distributed\(^-\)
CAS

- Tunable per container
- Cross cloud portability
- Cross SAN portability
- No special skills needed!
- Low latency
CAS feature comparison

**DAS**

**Benefits:**
- Simple
- Ties application to storage
- Predictable for capacity planning
- App deals with resiliency
- Can be faster

**Concerns:**
- Under-utilized hardware
  - 10% or less utilization
- Wastes data center
- Difficult to manage
- Lacks storage features
- Cannot be repurposed - made for one workload
- Does not support mobility of workloads via containers
- Cross cloud impossible

**OpenEBS = “CAS”**

- ✓ Simple
- ✓ No new skills required
- ✓ Per microservice storage policy
- ✓ Data protection & snapshots
- ✓ Reduces cloud vendor lock-in
- ✓ Eliminates storage vendor lock-in
- ✓ Highest possible efficiency
- ✓ Large & growing OSS community
- ✓ Natively cross cloud

**Distributed**

**Benefits:**
- Centralized management
- Greater density and efficiency
- Storage features such as:
  - Data protection
  - Snapshots for versioning

**Concerns:**
- Additional complexity
- Enormous blast radius
- Expensive
- Requires storage engineering
- Challenged by container dynamism
- No per microservice storage policy
- I/O blender impairs performance
- Locks customers into vendor
- Cross cloud impossible
Why CAS?

- Truly Cloud Native (Storage is a microservice)
- Enables cloud native blue-green deployment of storage (no need to upgrade all volumes at once)
- Highest granularity of storage policies (assume one storage controller handles only one type of workload)
CAS benefits

Cloud Native challenges
1. Workloads much smaller
2. Ephemeral duration
3. 10-100x increase in quantity
4. Mobile workloads
5. Immutable not tunable
6. Developers responsible for operations
7. Easiest solutions lead to cloud lock-in

CAS Answers
1. Keep data local
2. Run controller in the POD
3. 10-100x more controllers
4. Follow the workloads
5. Stateless control of state
6. Make storage just another microservice each team controls. No YASSS needed!
7. Cross cloud cMotion - thanks to OpenEBS & MayaOnline
CAS architecture example

- Each volume has a controller and three replica pods.
Blue Green Deployment example with CAS

<table>
<thead>
<tr>
<th>#</th>
<th>volume name</th>
<th>ctrl ver</th>
<th>rep1 ver</th>
<th>rep2 ver</th>
<th>rep3 ver</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>mysql-vol1</td>
<td>1.2</td>
<td>1.2</td>
<td>1.2</td>
<td>1.2</td>
</tr>
<tr>
<td>2</td>
<td>postgres-vol1</td>
<td>1.1</td>
<td>1.1</td>
<td>1.1</td>
<td>1.1</td>
</tr>
<tr>
<td>3</td>
<td>postgres-vol2</td>
<td>1.2</td>
<td>1.2</td>
<td>1.2</td>
<td>1.2</td>
</tr>
<tr>
<td>4</td>
<td>mongoDB-vol1</td>
<td>1.2</td>
<td>1.2</td>
<td>1.2</td>
<td>1.2</td>
</tr>
</tbody>
</table>
Blue Green Deployment example with CAS

<table>
<thead>
<tr>
<th>#</th>
<th>volume name</th>
<th>ctrl ver</th>
<th>rep1 ver</th>
<th>rep2 ver</th>
<th>rep3 ver</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>mysql-vol1</td>
<td>1.2</td>
<td>1.2</td>
<td>1.2</td>
<td>1.2</td>
</tr>
<tr>
<td>2</td>
<td>postgres-vol1</td>
<td>1.1</td>
<td>1.1</td>
<td>1.1</td>
<td>1.1</td>
</tr>
<tr>
<td>3</td>
<td>postgres-vol2</td>
<td>1.2</td>
<td>1.2</td>
<td>1.2</td>
<td>1.2</td>
</tr>
<tr>
<td>4</td>
<td>mongoDB-vol1</td>
<td>1.2</td>
<td>1.2</td>
<td>1.2</td>
<td>1.2</td>
</tr>
</tbody>
</table>
Blue Green deployment of mysql-vol1
Blue Green deployment of mysql-vol1

- mysql
- k8s service load balancer (target service)
- mysql-vol1-ctrl-pod 1.2
- mysql-vol1-repl-1-pod
- mysql-vol1-repl-2-pod
- mysql-vol1-repl-3-pod
- OpenEBS Maya Operator
  - switch data traffic to blue (1.3)
  - Point replicas to blue (1.3)
  - Analyze / observe the traffic
  - Retain if okay
  - Revert if not okay
Blue Green deployment of mysql-vol1

- Controller pod is upgraded to 1.3
- Upgrade repl-1 pod to 1.3 and observe (revert to 1.2 if needed)
- Rebuild repl-1
Blue Green deployment of mysql-vol1

1. Controller pod is upgraded to 1.3
2. repl-1 pod is upgraded to 1.3
3. Upgrade repl-2 pod to 1.3 and observe (revert to 1.2 if needed)
4. rebuild repl-2
Blue Green deployment of mysql-vol1

- Controller pod is upgraded to 1.3
- repl-1 pod is upgraded to 1.3
- repl-2 pod is upgraded to 1.3
- Upgrade repl-3 pod to 1.3 and observe (revert to 1.2 if needed)
- rebuild repl-3
Blue Green Deployment example with CAS

OpenEBS Maya Operator

mysql-vol1 is upgraded to 1.3
Now upgrade postgres-vol1 to 1.3

<table>
<thead>
<tr>
<th>#</th>
<th>volume name</th>
<th>ctrl ver</th>
<th>rep1 ver</th>
<th>rep2 ver</th>
<th>rep3 ver</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>mysql-vol1</td>
<td>1.3</td>
<td>1.3</td>
<td>1.3</td>
<td>1.3</td>
</tr>
<tr>
<td>2</td>
<td>postgres-vol1</td>
<td>1.1</td>
<td>1.1</td>
<td>1.1</td>
<td>1.1</td>
</tr>
<tr>
<td>3</td>
<td>postgres-vol2</td>
<td>1.2</td>
<td>1.2</td>
<td>1.2</td>
<td>1.2</td>
</tr>
<tr>
<td>4</td>
<td>mongoDB-vol1</td>
<td>1.2</td>
<td>1.2</td>
<td>1.2</td>
<td>1.2</td>
</tr>
</tbody>
</table>
Upgrade operations - B/G

- With CAS architecture, the upgrade process can be automated without human intervention. Something that is unheard of in the storage world.
What about speeds and feeds?

- The bottleneck is often now the Linux kernel
- User space I/O and networking is the future
  - You already know about SPDK, FD.io, VPP
  - You might find interesting:
    - https://github.com/openebs/vhost-user
  - We are looking for collaboration in this area as well
CAS - References

Blog published by CNCF on CAS

https://www.cncf.io/blog/2018/04/19/container-attached-storage-a-primer/
Q&A

Thank you !!

slack.openebs.io