KubeCon CloudNativeCon Europe 2026

#KubeCon #CloudNativeCon

Kubernetes Data Protection WG Deep Dive

Dave Smith-Uchida (Veeam) & Xing Yang (VMware by Broadcom)

Agenda

- Motivation
- Who are involved
- Projects
  - Backup Repository - COSI
  - Consistent Group Snapshot
  - Changed Block Tracking (CBT)
- Whitepaper
  - Best Practices for Kubernetes Applications for Data Protection
    - Deep dive on Operator data protection
- How to get involved

Motivation

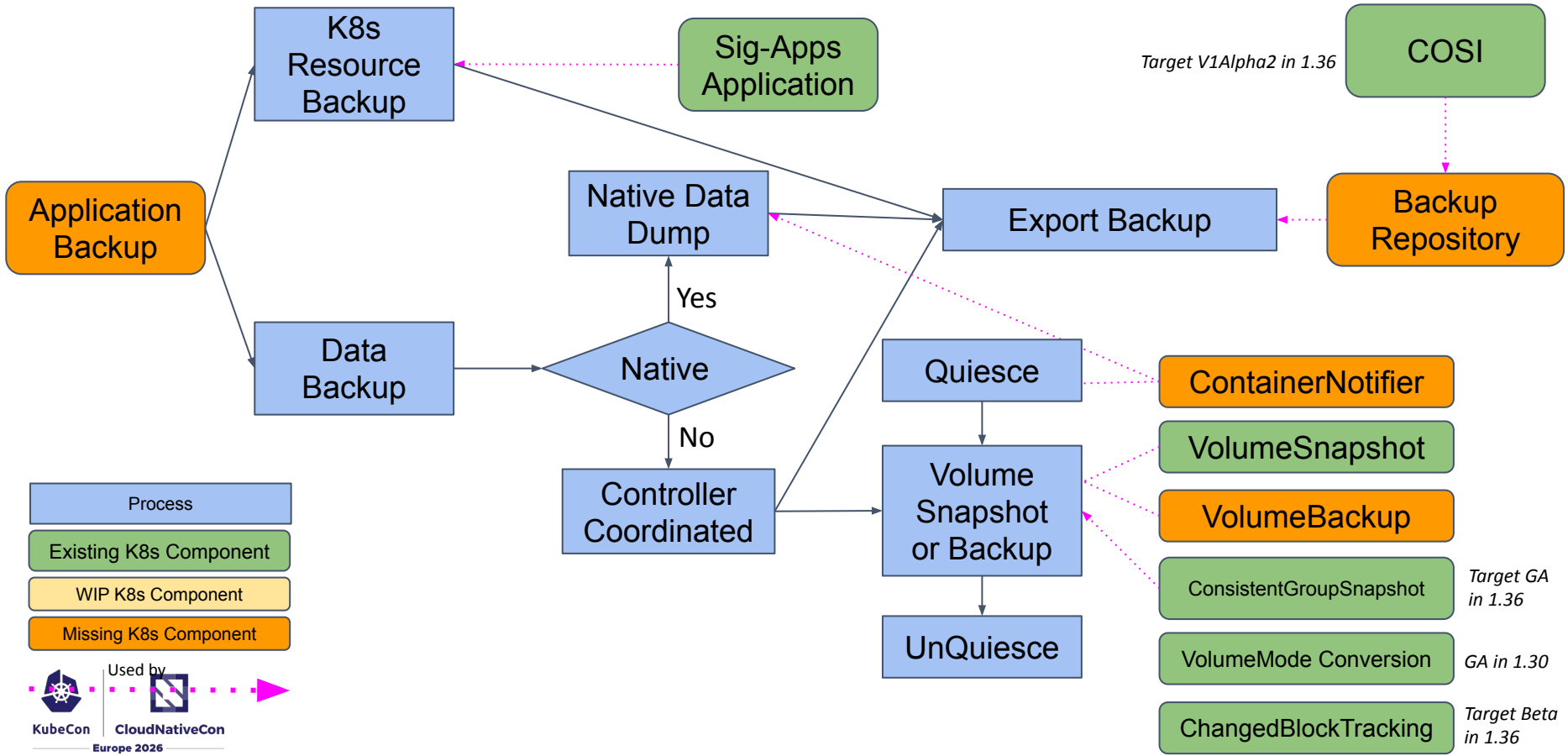
- Day-1 operations for stateful workloads are well supported
  - Persistent volume operations
  - Workload APIs (deployment/statefulset etc)
- More and more stateful workload are moving to K8s
  - Rapid iteration and agile deployment
  - Built in scalability
  - Portability
- Day-2 operations for data protection are still limited
  - Gitops - Flux/ArgoCD - But
    - Secret/ConfigMap and/or other sensitive data
    - Data stored in persistent volume

Who are involved

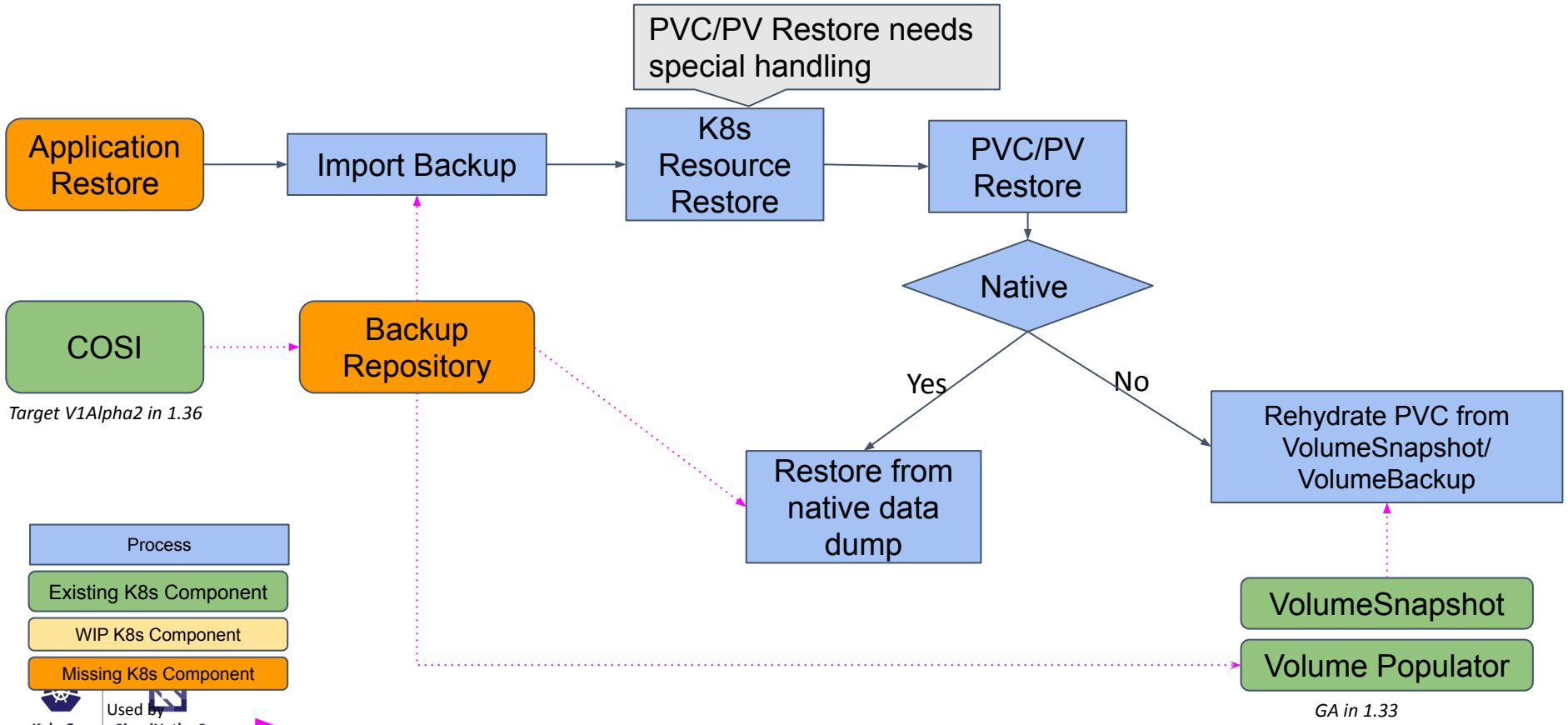
The following companies are supporting this initiative:

Arrikto, Catalogic Software, Cohesity (Veritas), Commvault, Dell EMC, Druva, Google, HPE, IBM, LINBIT, Linkedin, DataCore (MayaData), Microsoft, Mongo, NetApp, Pure Storage (Portworx), Red Hat, Rubrik, SUSE, Trilio, Veeam (Kasten), VMware by Broadcom

Backup


K8s Resource Backup
Sig-Apps Application
Target V1Alpha2 in 1.36
COSI
Application Backup
Native Data Dump
Export Backup
Backup Repository
Yes
Data Backup
Native
Quiesce
ContainerNotifier
No
Controller Coordinated
Volume Snapshot or Backup
VolumeSnapshot
VolumeBackup
ConsistentGroupSnapshot
Target GA in 1.36
UnQuiesce
VolumeMode Conversion
GA in 1.30
ChangedBlockTracking
Target Beta in 1.36
Process
Existing K8s Component
WIP K8s Component
Missing K8s Component
Used by
KubeCon
CloudNativeCon
Europe 2026

Restore

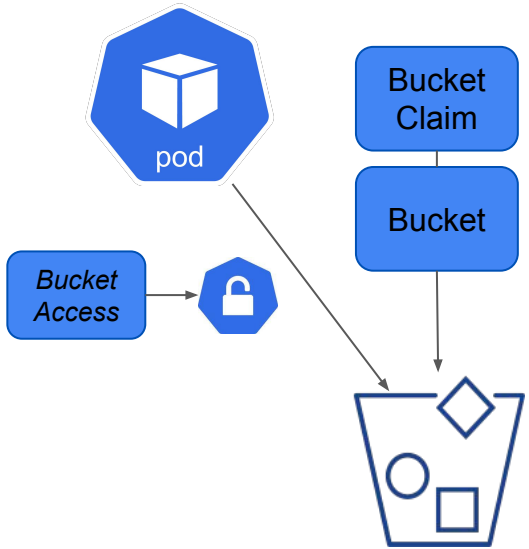


Backup Repository

- Container Object Storage Interface (COSI) provides a standard way for provisioning and consuming object storage in Kubernetes.
- It can be used to support a Backup Repository such as an object store.
- COSI Components
  - COSI Controlle: validates, authorizes and binds COSI created buckets to BucketClaims.
  - COSI Sidecar: watches COSI K8s API objects and calls COSI Driver.
  - COSI Driver: communicates with object storage providers to conduct bucket related operations.

Backup Repository

- COSI targets v1alpha2 in K8s 1.36
- COSI K8s APIs
  - Bucket, BucketClaim, BucketClass
  - BucketAccess, BucketAccessClass
- COSI gRPC interfaces for object storage providers to provision buckets
- Notable v1alpha2 changes
  - BucketAccesses can now request access to multiple BucketClaims
  - BucketAccessModes
    - 3 types of access: ReadWrite, WriteOnly, ReadOnly
    - 3 types of data: ObjectData, ObjectMetadata, BucketMetadata



Bucket, BucketClaim, and BucketClass

```yaml
apiVersion: v1alpha2
kind: BucketClass
metadata:
 name: bc1
deletionPolicy: delete
driverName: sample.cosi.io
parameters:
 key: value
```

```yaml
apiVersion: v1alpha2
kind: BucketClaim
metadata:
 name: bclaim1
 namespace: ns1
spec:
 bucketClassName: bc1
 protocols:
 - s3
status:
 conditions:
   provisioned: true
   resourceValidated: true
 boundBucketName: bcl-$uuid
 protocols:
 - s3
```

```yaml
apiVersion: v1alpha2
kind: Bucket
metadata:
 name: bcl-$uuid
spec:
 protocols:
 - s3
 parameters:
  key: value
 driverName: sample.cosi.io
 bucketClaimRef:
  name: bclaim1
  namespace: ns1
  uid: <xxxxxx>
 deletionPolicy: Retain
status:
 conditions:
   provisioned: true
   resourceValidated: true
 bucketId: <xxxxxx>
 protocols:
 - s3
```

BucketAccess and BucketAccessClass

```yaml
apiVersion: v1alpha2
kind: BucketAccessClass
metadata:
 name: bac1
parameters:
 key: value
authenticationType: KEY
driverName: sample.cosi.io
multipleBucketAccess:
MultipleBuckets # or SingleBucket
disabllowedBucketAccessModes:
  bucketMetaData:
  - ReadWrite
  - WriteOnly
```

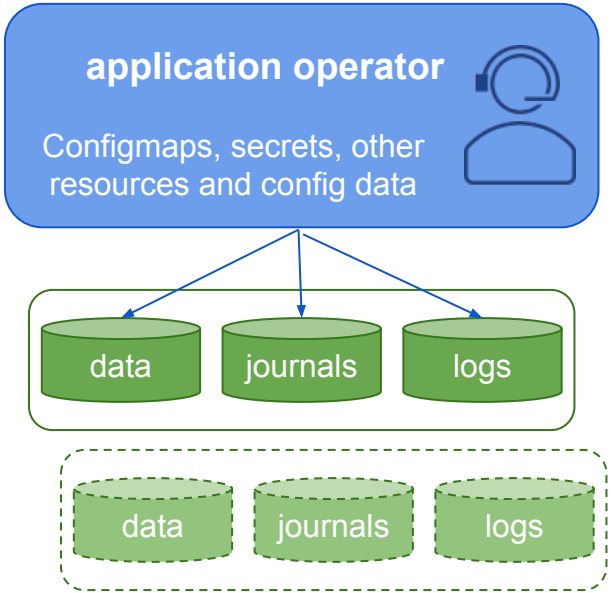
```yaml
apiVersion: v1alpha2
kind: BucketAccess
metadata:
 name: cosi-access
 namespace: ns1
spec:
 bucketAccessClassName: bac1
 bucketClaims:
 - name: bclaim1
   namespace: ns1
   uid: <xxxxxx>
 - name: bclaim2
   namespace: ns1
   uid: <xxxxxx>
 serviceAccountName: bucketcreds1
 protocol: s3
status:
  conditions:
    provisioned: true
    resourceValidated: true
  .....
```

User creates pod with a projected volume pointing to the secret in BucketAccess

```yaml
apiVersion: v1
kind: Pod
metadata:
 name: cosi-pod
spec:
 containers:
 - name: cosi-container
   image: myimage:1.0
   volumeMounts:
   - name: cosi-bucket
     mountPath: "/cosi/bucket1"
 volumes:
 - name: cosi-bucket
   projected:
     sources:
     - secret:
         name: bucketcreds1
```

Consistent Group Snapshot

- Support crash consistency when application consistency is not available or not practical
- Ensure write order consistency of multiple volumes in the same group
- Better performance



Consistent Group Snapshot - Design

- Targeting GA in K8s 1.36
- Kubernetes APIs
  - VolumeGroupSnapshot
  - VolumeGroupSnapshotContent
  - VolumeGroupSnapshotClass
- CSI spec changes
  - New Group Controller Service
  - New gRPC interfaces for create/delete/get volume group snapshot
- New controller logic in snapshot-controller and CSI snapshotter sidecar

VolumeGroupSnapshot APIs

```yaml
apiVersion: groupsnapshot.storage.k8s.io/v1
kind: VolumeGroupSnapshotClass
metadata:
 name: class1
deletionPolicy: delete
driver: sample.io
```

```yaml
apiVersion: groupsnapshot.storage.k8s.io/v1
kind: VolumeGroupSnapshot
metadata:
 name: gs1
 namespace: ns1
spec:
 volumeGroupSnapshotClassName: class1
 source:
   selector:
     matchLabels:
       groupsnapshot.storage.k8s.io/volumeGroupSnapshotName:gs1
status:
 boundVolumeGroupSnapshotContentName:
   groupsnapcontent-$uuid
 readyToUse: true
 ……..
```

```yaml
apiVersion: groupsnapshot.storage.k8s.io/v1
kind: VolumeGroupSnapshotContent
metadata:
 name: groupsnapcontent-$uuid
spec:
 volumeGroupSnapshotClassName: class1
 source:
   persistentVolumeNames:
    - pvc-ff202cbe-264a-48d3-9daf-bafebfcffffd
    - pvc-b26ab151-1800-46fc-a6f9-7f0f8c2be5ec
 driver: sample.io
 volumeGroupSnapshotRef:
  name: gs1
  namespace: ns1
  uid: <xxxxxx>
 deletionPolicy: Delete
status:
 volumeGroupSnapshotHandle:
   00b9732e-8c48-11ee-92fb-5e1359b14971
 readyToUse: true
 ……..
```

Apply label to PVCs to be snapshotted together

Use Group Snapshot for Restore

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: pvc0-restore
 namespace: ns1
spec:
 storageClassName: sc1
 dataSource:
   apiGroup: snapshot.storage.k8s.io
   name: snapshot-xxxxxxxx
   kind: VolumeSnapshot
 accessModes:
 - ReadWriteOnce
 resources:
   requets:
     storage: 1Gi
```

- A VolumeGroupSnapshot has an associated individual VolumeSnapshot for each original PVC
- To restore, create a new PVC from a VolumeSnapshot
- Repeat until all PVCs are restored.

Changed Block Tracking (CBT) API

- One set of interfaces
- Operates on snapshot(s) of the same persistent volume
- Efficiently identify changed blocks between two snapshots
- Incremental volume backup
- Decouple backup system from storage system

SnapshotMetadata CSI Specification

Storage providers should implement a new CSI service and RPCs in their CSI drivers to support CBT:

- New SnapshotMetadata Service
  - SNAPSHOT_METADATA_SERVICE Plugin Capability
  - Two new CSI RPCs
    - GetMetadataAllocated
    - GetMetadataDelta
- Supports block volume only

SnapshotMetadata in Kubernetes

Backup software should adopt these APIs to support CBT:

- Two Kubernetes SnapshotMetadata gRPC RPCs
  - Very similar to CSI SnapshotMetadata Service gRPC PRCs
    - GetMetadataAllocated
    - GetMetadataDelta
  - Consumes CBT via gRPC
  - The `SnapshotMetadata` sidecar container
- New SnapshotMetadataService CRD
  - Created by the CSI driver

- The CBT feature is targeting Beta in K8s 1.36 release.

KubeCon CloudNativeCon Europe 2026

Whitepaper

Whitepaper

Kubernetes Data Protection: Best Practices

Kubernetes now has mechanisms for data protection and multiple strategies for ensuring application resiliency including GitOps, backup/restore and remote replication

Protecting an application involves:

- Understanding the resiliency needs of the application
- Selecting the right strategy to meet the resiliency needs
- Making modifications, if necessary to the application to handle the strategy

Whitepaper is under construction, contributors/reviewers are needed!

Operator characteristics

- Manage applications based on Custom Resources
- Allocate Kubernetes resources for applications
- Handle application upgrade
- Sometimes manage backup/restore and/or replication of managed applications

Data Protection challenges with Operators

- Relationship of managed resources with controlling CR
- Determining order of creation of resources on restore
- Quiescing operators during backup/restore
- Using operator data protection operations
- Operators calling operators
- Backup/restore of the operator itself

Determining order of creation on restore

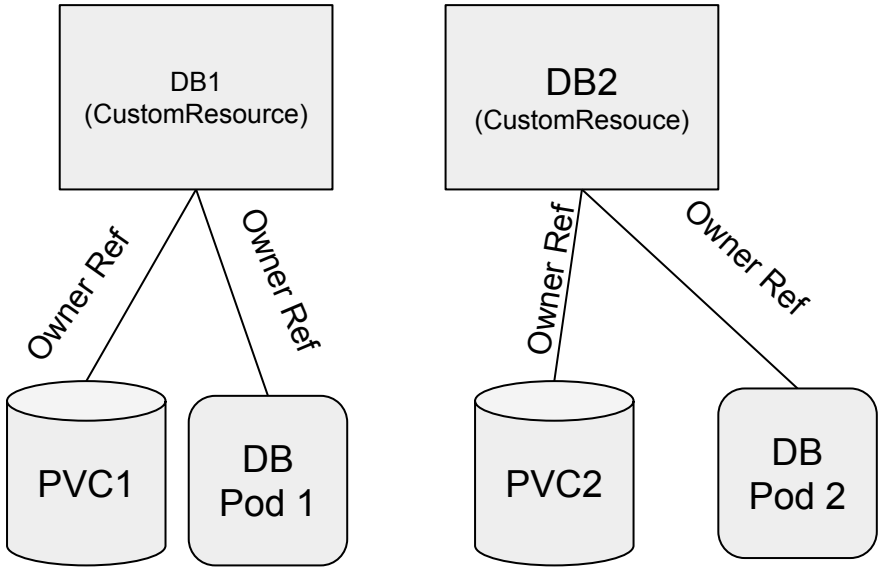
- Operators create resources in a top down manner
  - Custom Resource causes the creation of resources
- On restore, resources need to be created bottom up
- If the custom resource is created first, then the operator will begin creating resources for it, conflicting with resources being restored by the application
- How can we determine the hierarchy of resources?

Relationship of managed resources with controlling CR

Operators create resources to implement the application specified by a Custom Resource
Understanding the relationship between the Custom Resources and the resources that implement them is important for data protection

- Protection of individual applications
- Exclusion of resources that are protected by operator data protection actions

OwnerRefs are a good start, not all operators set them properly
Lots of heuristics and special cases for determining these relationships



Quiescing of operators

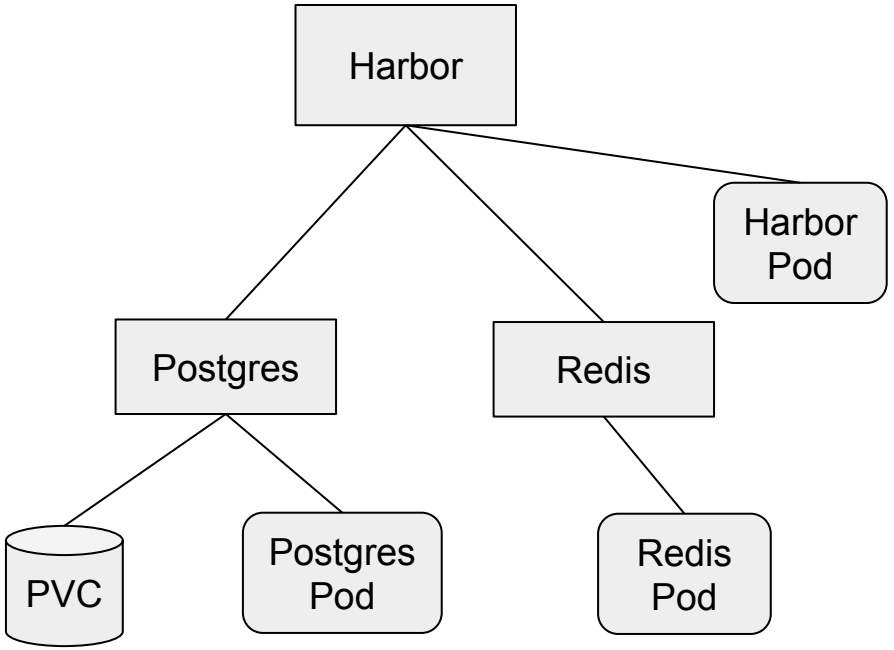
- Operators may make changes during backup phase that could cause the backup to be inconsistent
- During restore, operators may make changes to resources while they are being restored
- Quiescing is usually handled by a script executed by the backup/restore utility
- Having a standard way to quiesce operators would allow the backup/restore utility to keep the operator from making changes during the process

Using operator data protection operations

- No standard interfaces - scripts need to be written and executed
- Resources that the operator is protecting need to be excluded from regular backup/restore operations
  - Need to determine resource dependencies
- Controlling where backups are stored and lifecycle differs between operators

Operators calling operators

- Operators can use other operators
- Resource hierarchy becomes more complex
- Understanding resource dependencies becomes even more important



Backup/restore of the operator itself

- Potential operator state (usually kept in Kubernetes resources)
- Operators provide services to entire cluster - multiple resources in multiple namespaces may rely on the operator
- Dependency graph inverts when the operator provides backup/restore functionality
  - Operator must be running before it can do restores
- In operator calling operator scenarios, all operators must be present
- Potential version mismatches on restore

Where do we go from here?

- Best practices for operators part of white paper
- We encourage operator developers to consider backup/restore/replication scenarios
- More work to be done on the data protection application side to handle operators seamlessly

How to get involved

KubeCon CloudNativeCon Europe 2026

- Home page: https://github.com/kubernetes/community/tree/master/wg-data-protection
- Bi-weekly meeting on Wednesdays at 9am Pacific Time. Meeting recordings available on YouTube.
  - Agenda doc
- Mailing list: https://groups.google.com/a/kubernetes.io/g/wg-data-protection
- Slack channel: #wg-data-protection

KubeCon CloudNativeCon Europe 2026

Thank You