KubeCon | CloudNativeCon

Europe 2022

WELCOME TO VALENCIA

KubeCon CloudNativeCon Europe 2022

Kubernetes Data Protection WG Deep Dive

Xiangqian Yu, Google & Xing Yang, VMware

Data Protection WG Leads

Xing Yang, VMware
CNCF TAG Storage Co-Chair
K8S SIG-Storage Co-Chair
Data Protection WG Co-Chair
Slack: xyang
Github: xing-yang
Twitter: @2000Xyang

Xiangqian Yu, Google
Data Protection WG Co-Chair
Slack: xiangqian
Github: yuxiangqian

Agenda

- Motivation
- Who are involved
- Key Updates
- Deep Dive
  - Volume mode conversion
  - Volume populator
  - CBT
  - Backup repository: COSI
  - ContainterNotifier
  - Volume Groups
  - Application Snapshot and Backup
- 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
- Day-2 operations for data protection are still limited
  - Gitops

Who are involved

The following companies are supporting this initiative:

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

Key Updates

- White Paper
  - Authors: Antony Bett, Phuong Hoang, Prashanto Kochavara, Stephen Manley, Tom Manville, Ben Swartzlander, Dave Smith-Uchida, Xing Yang, Xiangqian Yu
- VolumeSnapshot v1beta1 API removal in K8s 1.24
  - KEP
- Previous Talks
  - 2021 North America, 2021 Europe, 2021 Asia
  - 2020 North America, 2020 Europe

Volume Mode Conversion - Motivation

- Why?
  - Allowing volume mode transition can introduce vulnerability to Kernel:
    - Block PVC -> Snapshot -> FileSystem PVC
  - Volume mode transition is needed for efficient backup workflow
    - File System PVC -> Snapshot -> Block PVC -> Block Diff

Volume Mode Conversion - How does it work?

- API Changes
  - A *SourceVolumeMode* field in *VolumeSnapshotContent*
  - An annotation *AllowVolumeModeChange* on *VolumeSnapshotContent*

- Behaviour
  - Reject volume mode change when rehydrating a volume from snapshot except the *AllowVolumeModeChange* annotation has been set to true.

Volume Mode Conversion - Status

- KEP 3141
- Blog PR
- Status
  - Alpha 1.24
- Dev Lead
  - Raunak Shah

Volume Populator - Motivation

- Why do we need volume populator
  - Create PVC from an external data source, not just PVC or VolumeSnapshot
  - Support WaitForFirstConsumer volume binding mode

Volume Populator Components

- A Volume Populator needs the following
  - A CRD it supports and can be specified in DataSourceRef of a PVC
  - A Volume Populator Controller: watches PVC with data sources it understands and handles it
- Kubernetes-csi built-in components
  - Lib-volume-populator: Volume Populator can use this library for K8s API level work. The logic for actually writing data into the volume based on a particular CR instance is left for the Volume Populator.
    - This repo includes a sample volume populator
  - volume data source validator: generates warning events on PVCs with data sources for which there is no populator.

Volume Populator - How does it work

- Enables the AnyVolumeDataSource feature gate: Beta in 1.24, enabled by default
- Deploys volume-data-source-validator controller
- Deploys Volume Populator
- Creates a CR that the populator understands
- Creates a PVC with data source pointing to that CR
- Volume Populator makes sure a PV is created and populated with data from that data source and binds with PVC

```yaml
apiVersion: hello.k8s.io/v1alpha1
kind: Hello
metadata:
  name: example-hello
spec:
  fileName: example.txt
  fileContents: Hello, world!
```

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: example-pvc
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 10Mi
  dataSourceRef:
    apiGroup: hello.k8s.io
    kind: Hello
    name: example-hello
  volumeMode: Filesystem
```

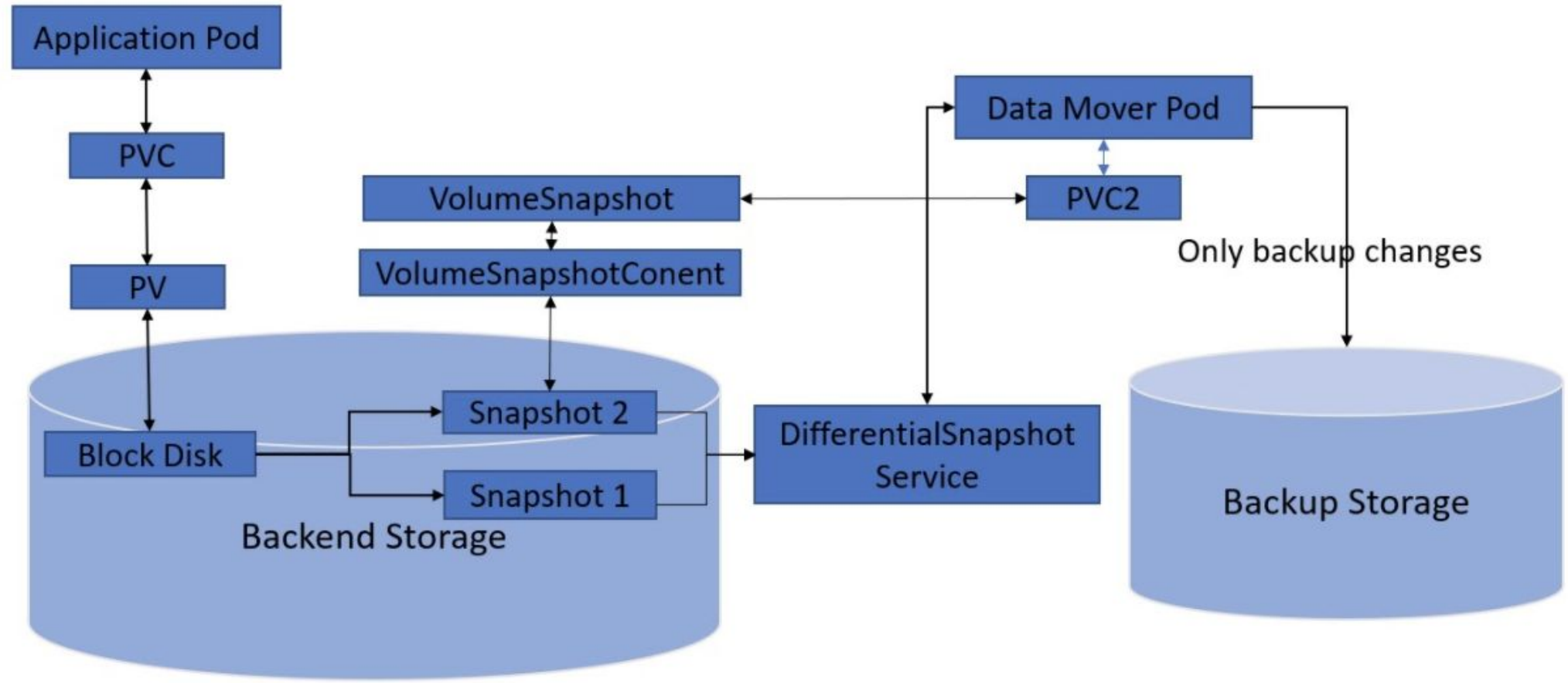
Volume Populator - Status

- Dev Lead: Ben Swartzlander
- Status
  - AnyVolumeDataSource alpha feature introduced in 1.18
  - Re-designed in 1.22
  - Moved to beta in 1.24
- References:
  - Volume populator KEP
  - Alpha Blog
  - WIP Beta Blog
  - Repos
    - https://github.com/kubernetes-csi/lib-volume-populator
    - https://github.com/kubernetes-csi/volume-data-source-validator

CBT - Motivation

- What is CBT?
  - Change Block Tracking: identifies blocks of data that have changed.
- Why CBT?
  - Backup: Extract CBT
    - Full backup is not space efficient, time consuming, and requires more bandwidth
  - Snapshot based replication
  - Alternatives to CBT
    - Full backups
    - Call each storage API to retrieve CBT directly

CBT - Design (WIP)



[Data Protection Workflows White Paper]

CBT - Design (WIP)

- Kubernetes API Object ChangedBlocks: https://github.com/phuongatemc/diffsnapcontroller
- CSI spec changes
  - A new capability or a new optional CSI service for differential snapshots
  - New CSI RPC: GetChangeBlocks
- API Aggregation to address performance concerns

CBT - Design (WIP)


```
type ChangedBlocksSpec struct {
        // If SnapshotBase is not specified, return all used blocks.
        SnapshotBase   string `json:"snapshotBase,omitempty"` // Snapshot handle, optional.
        SnapshotTarget string `json:"snapshotTarget"`        // Snapshot handle, required.
        VolumeId       string `json:"volumeId,omitempty"`    // optional
        StartOffset    string `json:"startOffset,omitempty"` // Logical offset from beginning of disk/volume.
        // Use string instead of uint64 to give vendor
        // the flexibility of implementing it either
        // string "token" or a number.
        MaxEntries uint64           `json:"maxEntries"`          // Maximum number of entries in the response
        Secrets    map[string]string `json:"secrets,omitempty"`   // Secrets required by Vendor to access snapshots.  Optional.
        Parameters map[string]string `json:"parameters,omitempty"` // Vendor specific parameters passed in as opaque key-value pairs.
        // Optional.
}
```

CBT - Design (WIP)


```
type ChangedBlocksStatus struct {
        State          string        `json:"state"`
        Error          string        `json:"error,omitempty"`
        ChangeBlockList []ChangedBlock `json:"changeBlockList"`      //array of ChangedBlock
        NextOffset      string        `json:"nextOffset,omitempty"` // StartOffset of the next “page”.
        VolumeSize      uint64        `json:"volumeSize"`          // size of volume in bytes
        Timeout         uint64        `json:"timeout"`             //second since epoch
}
```

```
type ChangedBlock struct {
        Offset  uint64 `json:"offset"`          // logical offset
        Size    uint64 `json:"size"`            // size of the block data
        Context []byte `json:"context,omitempty"` // additional vendor specific info.  Optional.
        ZeroOut bool   `json:"zeroOut"`  // If ZeroOut is true, this block in SnapshotTarget is zero out.
        // This is for optimization to avoid data mover to transfer zero blocks.
        // Not all vendors support this zeroout.
}
```

CBT - Status

- Dev Lead: Phuong N. Hoang
- Status
  - Design and POC in progress
- References:
  - POC repo
  - WIP KEP
  - CBT meeting minutes

Backup Repository - Motivation

- Why do we need a backup repository
  - Need a location or repo to store metadata from K8 clusters
  - Need to store data somewhere else as local snapshots provide single point of failure
  - Data explosion - need an intelligent way to store backups
- Different types of backup repositories

Backup Repository - COSI

- COSI Components
  - COSI ControllerManager: 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.
- COSI K8s APIs
  - Bucket
  - BucketClaim
  - BucketAccess
  - BucketClass
  - BucketAccessClass
- COSI gRPC interfaces for object storage providers to provision buckets

COSI - Status

- Dev Lead: Sidhartha Mani
- Status
  - KEP review in progress
  - SIG-Storage subproject kubernetes-cosi
  - Weekly design review meetings
  - Target Alpha in 1.25
- References
  - KEP in review
  - COSI repos

Quiesce and Unquiesce Hooks - Motivation

- Why do we need quiesce and unquiesce hooks?
  - Quiesce application before taking a snapshot and unquiesce afterwards to ensure application consistency
- Different workloads have different semantics

Quiesce and Unquiesce Hooks - Design

- Quiesce and unquiesce hooks proposal (design in progress)
  - KEP in review: ContainerNotifier
  - ContainerNotifier supports general use cases beyond quiesce and unquiesce
  - Provides a generic mechanism to run commands in containers but application specific semantics is out of scope

ContainerNotifier KEP

1. Inline ContainerNotifier list in the *Container* core type.
2. Inline type ContainerNotifierHandler defines a command.
3. Creating PodNotification to request triggering corresponding ContainerNotifiers.
4. Results recorded in each ContainerNotificationStatus.
5. Providing pod level aggregation.

```
Type Container struct {
        Notifiers []ContainerNotifier
}
Type ContainerNotifier struct {
        Name string
        Handler *ContainerNotifierHandler
        TimeoutSeconds int32
}
Type PodNotificationSpec struct {
        PodName string
        Notifier string
        TTLSecondsAfterCompletion *int32
}
Type PodNotificationStatus struct {
        UID types.UID
        Containers []ContainerNotificationStatus
        StartTime *metav1.Time
        CompleteTime *metav1.Time
        State PodNotificationStateType
        Error *PodNotificationError
}
```

ContainerNotifier KEP (cont.)

Introduces Notification type in phase 2

1. High level aggregates in status
2. Policy to control pod selection behavior
3. User-friendly, especially for signal use case

```
Type NotificationSpec struct {
        Selector *metav1.LabelSelector
         // supports AllPods and PreExistingPodsOnly
        Policy *PodSelectionPolicy
        Notifier string // name of the ContainerNotifier
        Parallelism int
        TTLSecondsAfterCompletion *int32
}

Type NotificationStatus struct {
        FailedCount int
        SucceededCount int
        StartTime *metav1.Time
        CompleteTime *metav1.Time
}
```

ContainerNotifier - Status

- Dev Lead: Xing Yang & Xiangqian Yu
- Status
  - KEP in review

Consistent Group Snapshot

- Why do we need 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
- Volume group and group snapshot
  - Design in progress
  - KEP in review
- Dev Lead: Xing Yang

Application Snapshot and Backup

- Why do we need application snapshot and backup?
  - To protect a stateful application
- Application snapshot and backup proposal (design in progress)
  - KEP
  - Snapshot and backup individual applications
- Dev Lead: Xing Yang & Xiangqian Yu

How to get involved

- 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/forum/#!forum/kubernetes-data-protection
- Slack channel: #wg-data-protection