KubeCon CloudNativeCon

Europe 2018

Policy-based Volume Snapshot Automation

Jing Xu (jinxu@google.com)
Anthony Yeh (enisoc@google.com)
Google Software Engineers

Outline

- Background
  - Snapshots, cloud volume snapshots, Kubernetes Volumes
- Kubernetes Volume Snapshot Design
  - VolumeSnapshot/VolumeSnapshotData API
- Metacontroller and Snapshot Policy
  - Snapshot policy controller
- Other Related Work
  - Restore workflow

It’s time to take a Snapshot!

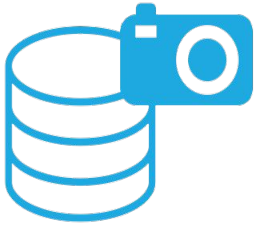
“Data is a precious thing and will last longer than the systems themselves” –Tim Berners-Lee

However,

- Data might be corrupted, deleted
- Disk might fail
- Software might malfunction

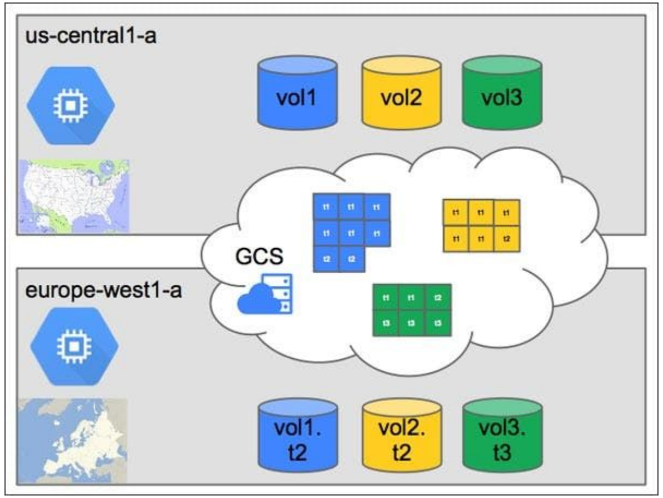
Don’t panic, take a snapshot

- A read-only copy of the data set frozen at a point in time
- Very little performance impact, less capacity than clone



Cloud Volume Snapshot

- Cloud Volume
  - Hard-disk-based volume that the underlying data is stored in the cloud
  - Disk can be attached/detached to VM instances
- Cloud Volume Snapshot
  - Fast: incremental snapshots
  - Available: single zone or global
- Snapshot is useful for
  - data protection
  - data replication among different zones



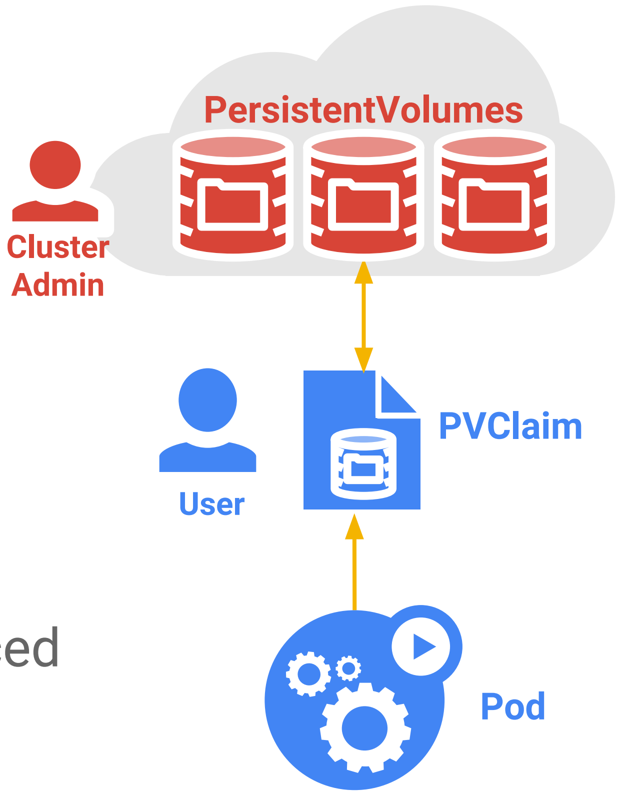
Quick Glance of Kubernetes Volumes

PersistentVolumeClaim (PVC) API Object

- Request for storage by a user, name-spaced
- Pods reference claims

PersistentVolume (PV) API Object

- Detailed storage information, non-namespaced
- Lifecycle independent of any individual pod



Google

PVC and PV Example

User ***$kubectl create -f pvc.yaml***

```
apiVersion:v1
kind: PersistentVolumeClaim
metadata:
  name: pd-claim
  namespace: default
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 50Gi
```

create

$kubectl describe pvc pd-claim

```
Name:            pd-claim
Namespace:       default
StorageClass:    standard
Status:          Bound
Volume:          pvc-305eb9fa-4349-11e8-a284
Annotations:     pv.kubernetes.io/bind-completed=yes
                 pv.kubernetes.io/bound-by-controller=yes
Capacity:        50G
Access Modes:    RWO
```

bind

$kubectl describe pv pvc-305eb9fa-4349-11e8-a284

```
Name:            pvc-305eb9fa-4349-11e8-a284
Labels:          failure-domain.kubernetes.io/zone=us-central1-b
StorageClass:    standard
Status:          Bound
Claim:           default/pd-claim
Reclaim Policy:  Delete
Capacity:        50Gi
Source:
    Type:        GCEPersistentDisk
    PDName:      e2e-test-jinxu-dynamic-pvc-305eb9fa-4349-11e8
    FSType:      ext4
```

Outline

- Background
  - Snapshots, cloud volume snapshots, Kubernetes Volumes
- Kubernetes Volume Snapshot Design
  - VolumeSnapshot/VolumeSnapshotData API
- Metacontroller and Snapshot Policy
  - Snapshot policy controller
- Other Related Work
  - Restore workflow

Snapshot workflow

- Create Snapshot
  - Types
    - Random: no application or file system interaction (***Supported***)
    - File system crash-consistent: freeze all of the I/O to the file system.
    - Application-aware: pause application, flush disk, unmount disks
  - Phases
    - Creating
      - point-in-time snapshot is created immediately
      - After snapshot is cut, application could be resumed safely.
    - Uploading (only for cloud providers)
      - Takes time to copy snapshot blocks to storage
      - First snapshot contains all data, the following are incremental

Snapshot workflow

- Use Snapshot
  - Create new volumes from snapshot
    - Data is populated to the new volume from the snapshot
  - New volumes from snapshot could be
    - Different configurations
      - Standard, ssd, ...
    - Different size
      - equal or bigger than original size
- Delete Snapshot
  - To definitively delete data from your snapshots, delete all snapshots.

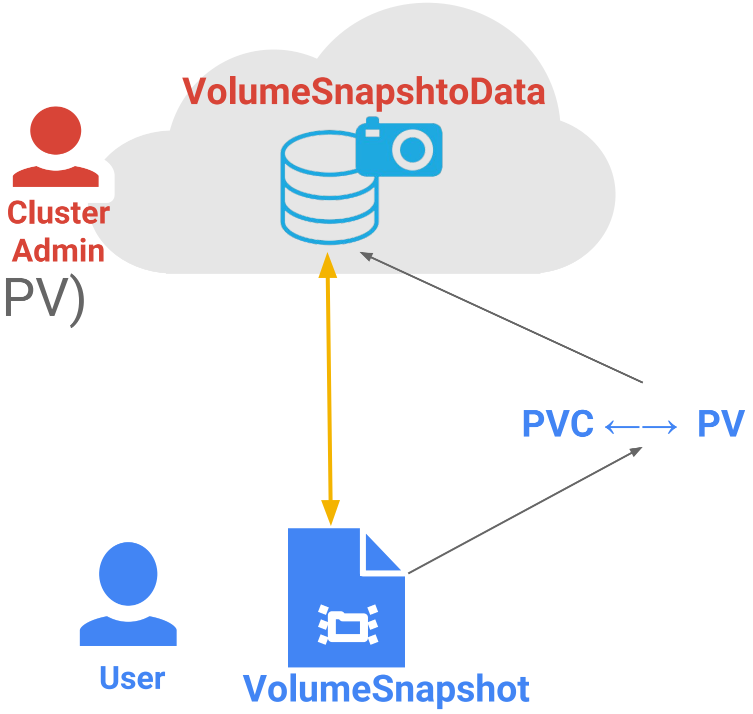
Kubernetes Volume Snapshots

VolumeSnapshot API Object

- Request for snapshot to a volume (PVC/PV)
- Name-spaced API object

VolumeSnapshotData API Object

- Detailed volume snapshot information
- Non-namespaced API object
- Lifecycle independent of any PVC/PV



Google

Create Snapshot

User***$kubectl create -f snapshot.yaml***

```
apiVersion:
volumesnapshot.external-storage.k8s.io/v1
kind:  VolumeSnapshot
metadata:
  name: snapshot-pd
spec:
   persistentVolumeClaimName:  pd-claim
```

create

$kubectl describe volumesnapshot snapshot-pd

```
Name:  snapshot-pd
Namespace:  default
Kind:  VolumeSnapshot
Metadata:
  Creation Timestamp:  2018-04-03T21:16:56Z
Spec:
  Persistent Volume Claim Name:  pd-claim
  Snapshot Data Name:  k8s-volume-snapshot-fd589156-428d
Status:
    Last Transition Time:  2018-04-03T21:17:07Z
    Status:              True
    Tyr                  Ready
```

$kubectl describe volumesnapshotdata

```
Name:              k8s-volume-snapshot-fd589156-428d
Kind:              VolumeSnapshotData
Metadata:
  Creation Timestamp:   2018-04-16T20:08:59Z
Spec:
  Gce Persistent Disk:
    Snapshot Id:   pv0115239093388465462б
  Persistent Volume Ref: pvc-305eb9fa-4349-11e8-a284
  Volume Snapshot Ref:
    Kind:  VolumeSnapshot
    Name:  default/snapshot-pd
```

Google

Create Volume From Snap

User`$kubectl create -f pvc.yaml`

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pd-claim-from-snap
  namespace: default
Annotations:
snapshot.alpha.kubernetes.io/snap
shot: snapshot-pd
spec:
  accessModes:
    - ReadWriteOnce
  resources:
     requests:
       storage: 100Gi
storageClassName: snapshot-promoter
```

create

`$kubectl describe pvc pd-claim-from-snap`

```
Name:            pd-claim-from-snap
Namespace:       default
StorageClass:    standard
Status:          Bound
Volume:          pvc-d1027af2-3e73-11e8-a284-42010a8
Annotations:     pv.kubernetes.io/bind-completed=yes
                 pv.kubernetes.io/bound-by-controller=yes
Capacity:        100G
Access Modes:    RWO
```

bind

`$kubectl describe pv pvc-d1027af2-3e73-11e8-a284-42010a8`

```
Name:            pvc-d1027af2-3e73-11e8-a284-42010a8
Labels:          failure-domain.beta.kubernetes.io/zone=us-central1-b
StorageClass:      standard
Status:            Bound
Claim:             default/pd-claim
Reclaim Policy:    Delete
Capacity:          100G
Source:
    Type:        GCEPersistentDisk
    PDName:      e2e-test-jinxu-dynamic-pvc-d1027af2-3e73-11e8
    FSType:      ext4
```

Data is populated into the volume

Google

Implementation

- External-storage
  - Custom Resources
    - VolumeSnapshot, VolumeSnapshotData API objects
  - Snapshot Controller and Provisioner
    - Control loop to create/delete, bind snapshot objects
    - External provision to create volume from snapshot

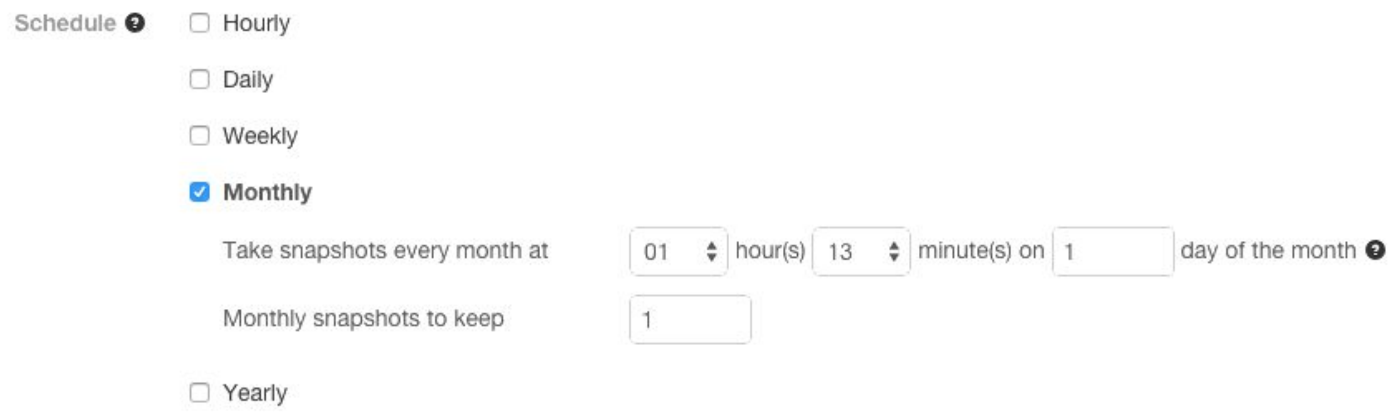
- Propose Alpha Version
  - In-tree APIs
  - CSI snapshot support

Outline

- Background
  - Snapshots, cloud volume snapshots, Kubernetes Volumes
- Kubernetes Volume Snapshot Design
  - VolumeSnapshot/VolumeSnapshotData
- Metacontroller and Snapshot Policy
  - Snapshot policy controller
- Other Snapshot Work
  - Restore workflow

Manage Many Snapshots

- How about a snapshot schedule?
  - Take snapshot periodically
  - Delete snapshot after it passes expiration date automatically



Schedule

- [ ] Hourly
- [ ] Daily
- [ ] Weekly
- [x] **Monthly**

Take snapshots every month at 01 hour(s) 13 minute(s) on 1 day of the month

Monthly snapshots to keep 1

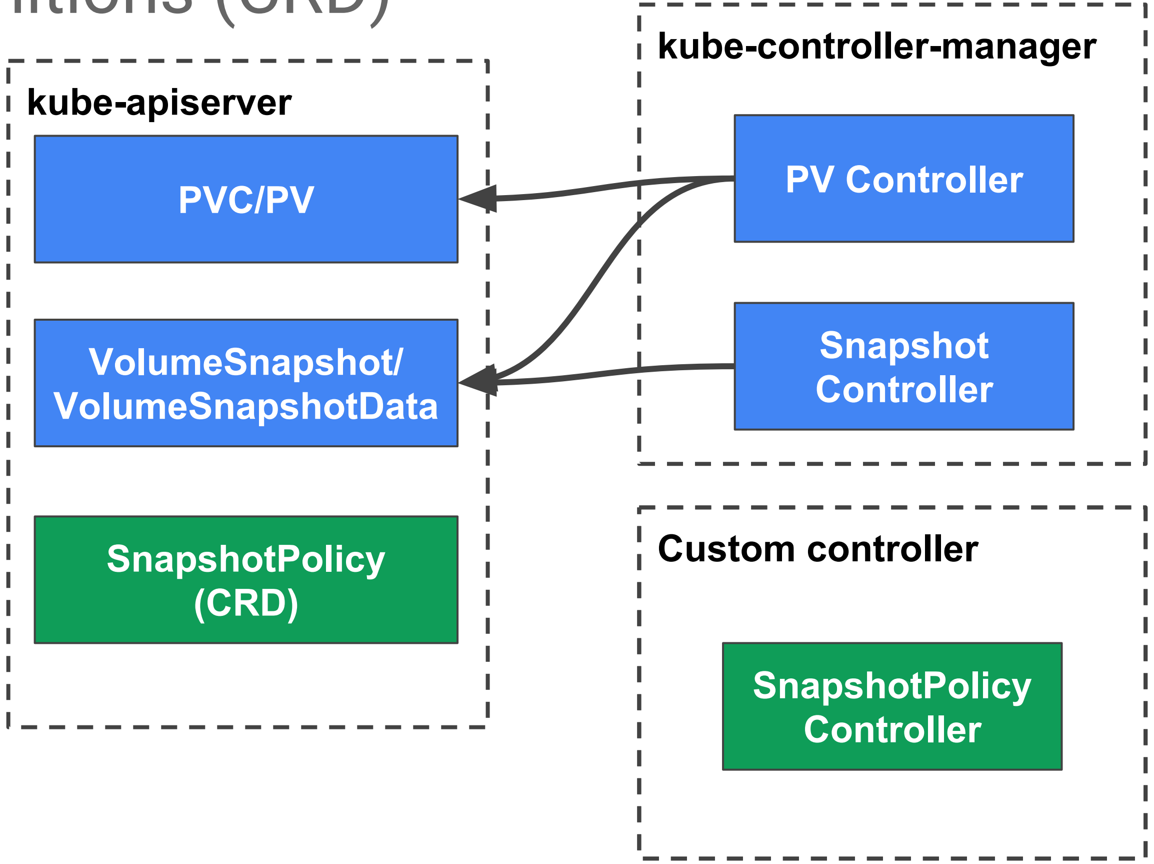
- [ ] Yearly

- Need Snapshot Policy
  - Controller can create/delete snapshots based on policy

Google

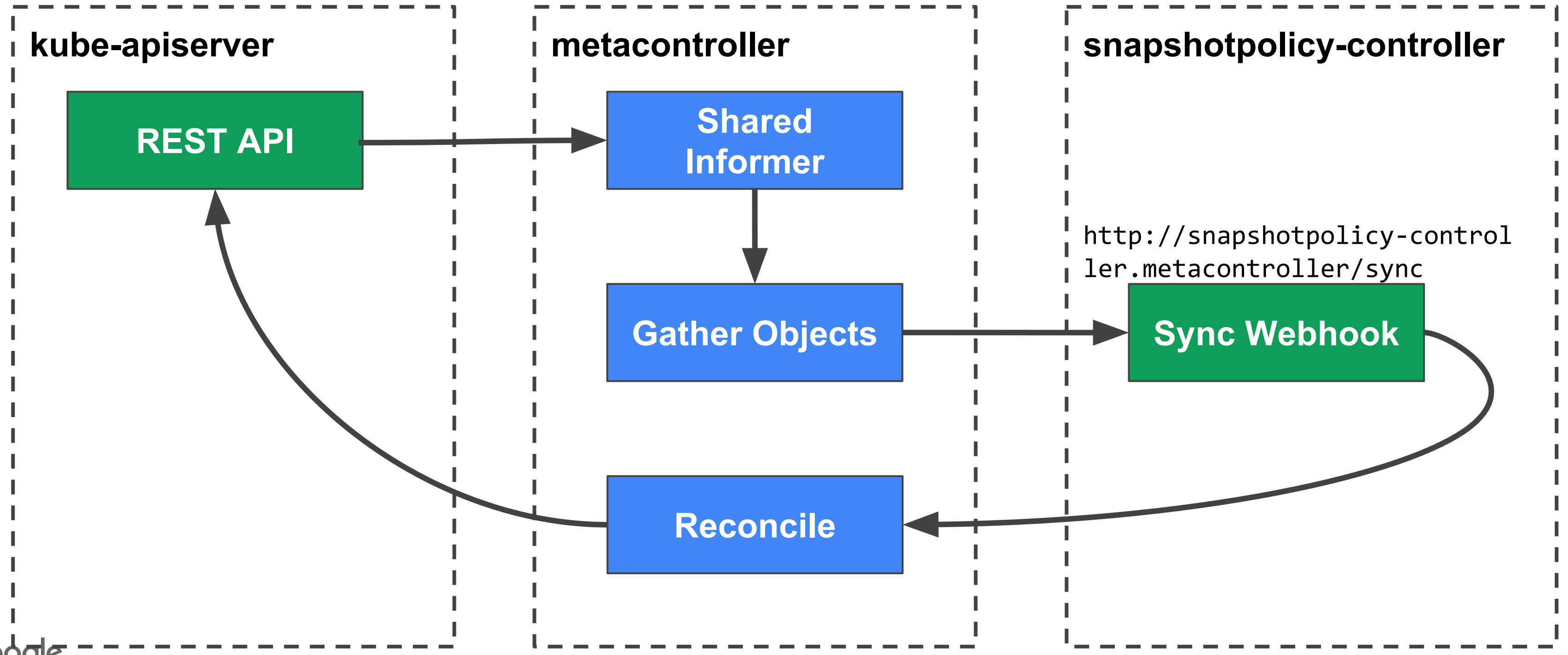
CustomResourceDefinitions (CRD)

- Your own defined object type similar to a native K8s API
- Snapshot Policy API to define snapshot schedule
- Also need controller to manage its behavior



Metacontroller

make it easy to define behavior for a new API or add custom behavior to existing APIs.



Snapshot Policy CRD

Dynamic configure policy and apply

```
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
  name: snapshotpolicies.ctl.k8s.io.com
spec:
  group: ctl.k8s.io.com
  version: v1
  scope: Namespaced
  names:
    plural: snapshotpolicies
    singular: snapshotpolicy
    kind: Snapshotpolicy
```

```
apiVersion: ctl.k8s.io.com/v1
kind: SnapshotPolicy
metadata:
  name: policy-1
spec:
  snapshotInterval: 6h
  validPeriod: 10d
  selector:
    matchLabels:
      app: snapshotpolicy-controller
```

Google

Snapshot Controller

Snapshotpolicy-controller sync hook

```
Sync() {
     For each snapshot
          If since(snapshot.creationTime)>validationPeriod
               Delete snapshot


     For each PVC
          If since(last_snapshot.creationTime) > snapshotInterval
               Create a new snapshot for the PVC
}
```

```
apiVersion: metacontroll
kind: CompositeControlle
metadata:
  name: snapshotpolicy-c
spec:
  parentResource:
    apiVersion: ctl.k8s
    resource: snapshotpo
  childResources:
    - apiVersion: v1
      resource: persiste
    - apiVersion: volume
      resource: volumesn
  ResyncPeriodSeconds: 5
  hooks:
    sync:
      webhook:
        url: http://snapshotpolicy-controller.metacontroller/sync
```

Google

A Simple Demo

Customize Your Own Policy

- Easy to change and apply different configurations
  - Modify policy file and apply it
- Customize business logic
  - Trigger a job to prepare application before taking a snapshot
  - Resume applications after snapshot is created
- Manage multiple policies
  - Match parent and children labels

Google

Outline

- Background
  - Snapshots, cloud volume snapshots, Kubernetes Volumes
- Kubernetes Volume Snapshot Design
  - VolumeSnapshot/VolumeSnapshotData
- Metacontroller and Snapshot Policy
  - Snapshot policy controller
- Other Snapshot Work
  - Restore workflow

Google

Create Volume From Snapshot

To create a new volume from snapshot

a. Adding a *snapshot source* in PVC yaml

b. A disk is created from snapshot

c. A new PVC and new PV bind

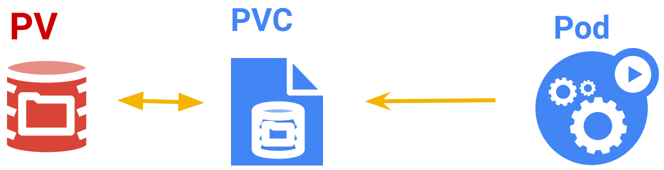
However,

- Need to modify PVC source file.
- Cannot work directly on PVC that is in-use. *(delete Pods, delete PVC, and then provision volume and create new PVC/PV)*

$kubectl create -f pvc.yaml

pvc.yaml

```
apiVersion: v1
Kind: PersistentVolumeClaim
Metadata:
Name  snapshot-data-claim
  namespace: default
Annotations:
snapshot.alpha.kubernetes.io/snapshot:
  Snapshot-pd
Spec:
  accessModes:
    - ReadWriteOnce
  Resources:
    Requests:
      Storage: 100Gi
```



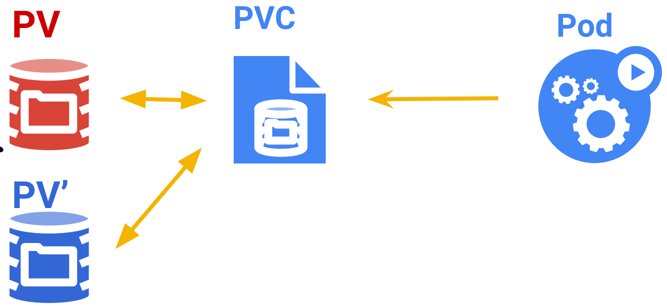
In-place Restore Volume from Snapshot

Restore Volume from snapshot

- When restore request *(restoreVolume API)* comes in, a new volume is provisioned from snapshot and a new PV is created
- When PVC is not in use (Pods are killed), bind the PVC to the new PV. The old PV is released or deleted

```
apiVersion: v1
kind: RestoreVolume
metadata:
  name: restore-claim
spec:
  snapshotSource: mysql-snapshot
  persistentVolumeClaimName: pd-claim
  oldVolumeReclaimPolicy: keep
```

Minimize Pod downtime
- time to switch the PVC bind pointer



Summary

Currently

- Volume Snapshot Out-of-tree: external storage repo
- Functions: volume snapshot create/delete, create volume from snapshot
- Plugins: GCE PD, AWS EBS, OpenStack, GlusterFS, HostPath

In the near future

- CSI volume snapshot support: Snapshot CSI Spec
- Metacontroller: SnapshotPolicy Controller
- Better Volume Snapshot Support: In-place Restore, in-tree VolumeSnapshot...

Collaborate!

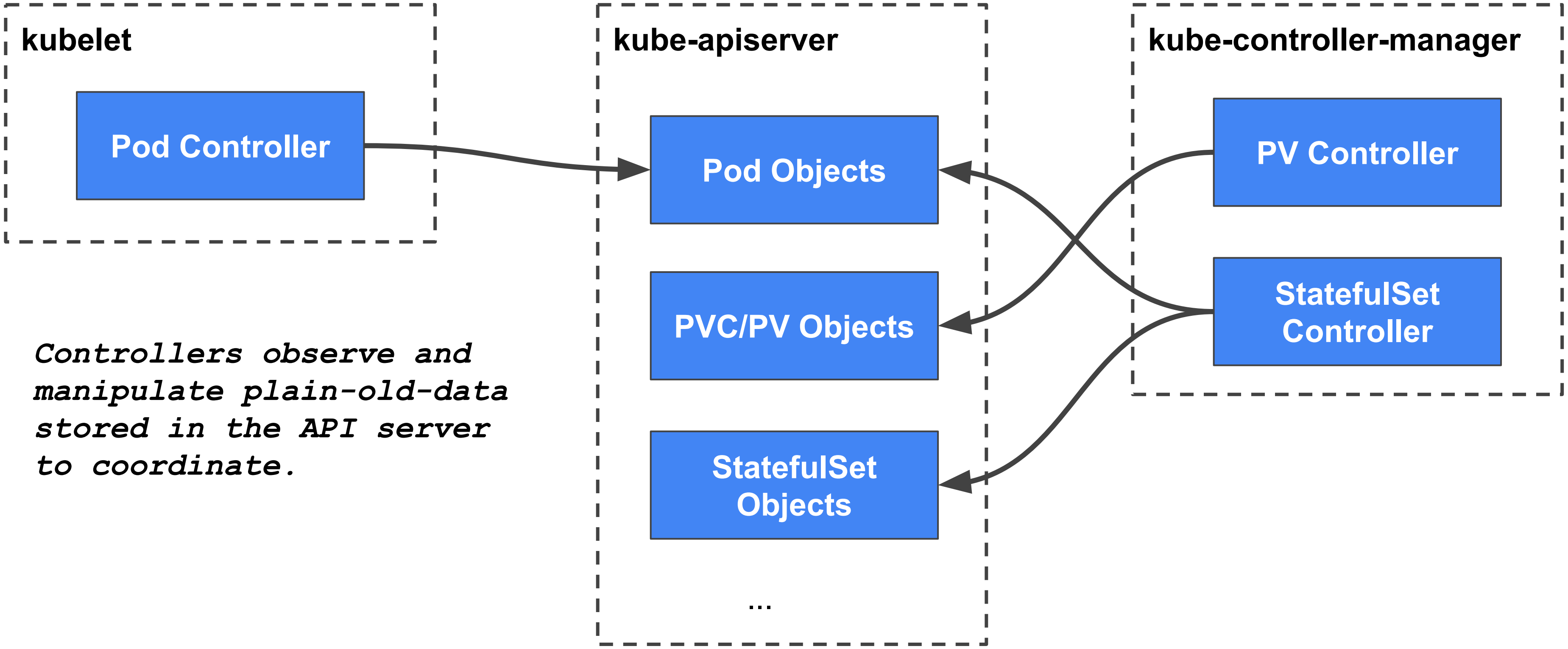
- (Google) Jing Xu, Anthony Yeh, and gke-storage-lifecycle team
- (OpenSDS) Xing Yang
- (RedHat) Huamin Chen, Thomas Smetana

get involved

Contact us with questions and feedback!

- Github: jingxu97 & xing-yang
- Slack: jinxu & xyang

Kubernetes API Objects and Controllers



Controllers observe and manipulate plain-old-data stored in the API server to coordinate.

Google

Volume Snapshot in Kubernetes

- Reasons for two-object model
  - Prevent other users copying VolumeSnapshot to other namespaces and using the snapshot
  - Hide security sensitive information (storage access keys etc.) from users

Volume Snapshot in Kubernetes

- Snapshot API objects
  - VolumeSnapshot (namespaced)
    - Spec: **PersistentVolumeClaimName, VolumeSnapshotDataName** (binding)
    - State: list of **VolumeSnapshotCondition,** each condition could be
      - **Uploading**
        - False: request is sent out, condition is unknown
        - True: snapshot is created, waiting for data to be copied to storage (finishes creating phase)
      - **Ready**
        - True: snapshot is ready to use (finishes uploading phase)
        - False: creation returns error
  - VolumeSnapshotData (non-namespaced)
    - Spec:
      - **VolumeSnapshotDataSource:** snapshot id information
      - **VolumeSnapshotRef:** bind VolumeSnapshot for references
      - **PersistentVolumeRef:** PV reference (used for restore volume provisioning)
    - State: no need

VolumeSnapshot

```
type VolumeSnapshot struct {
        metav1.TypeMeta `json:",inline"`
        Metadata        metav1.ObjectMeta `json:"metadata"`

        // Spec represents the desired state of the snapshot
        Spec VolumeSnapshotSpec

        // Status represents the latest observer state of the snapshot
        Status VolumeSnapshotStatus
}
```

```
// VolumeSnapshotSpec is the desired state of the volume snapshot
type VolumeSnapshotSpec struct {
        // the name of the PVC being snapshotted
        PersistentVolumeClaimName string

        // the name of VolumeSnapshotData binds to VolumeSnapshot
        SnapshotDataName string
}
```

```
// VolumeSnapshotStatus is the status of the VolumeSnapshot
type VolumeSnapshotStatus struct {
        // The time the snapshot was successfully created
        CreationTimestamp metav1.Time

        // Represent the latest available observations about the snapshot
        Conditions []VolumeSnapshotCondition
}
```

VolumeSnapshotData

```
type VolumeSnapshotData struct {
        metav1.TypeMeta `json:",inline"`
        // +optional
        Metadata metav1.ObjectMeta `json:"metadata"`

        // Spec represents the desired state of the snapshot
        // +optional
        Spec VolumeSnapshotDataSpec

}
```

```
type VolumeSnapshotDataSpec struct {
        // Source represents the location and type of the volume snapshot
        VolumeSnapshotDataSource

        // binding between VolumeSnapshot and VolumeSnapshotData
        // +optional
        VolumeSnapshotRef *core_v1.ObjectReference

        // represents the PersistentVolume that the snapshot has been
        // taken from
        // +optional
        PersistentVolumeRef *core_v1.ObjectReference
}
```

```
type AWSElasticBlockStoreVolumeSnapshotSource struct {

        // Unique id of the persistent disk snapshot resource.

        SnapshotID string `json:"snapshotId"`

}
```