

Volumes	2
Persistent Volume	2
Provisioning	3
Static	3
Dynamic	3
Binding	4
Storage object in use protection	4
Reclaim	4
Retain	4
Delete	4
Recycle	5
Expanding PVC	5
Persistent Volume Spec	6
Capacity	7
Access Mode	7
Class	7
Reclaim Policy	7
Phase	7
Node Affinity	8
Persistent Volume Claim Spec	8
Access Mode	8
VolumeMode	8
Resources	9
Class	9
Selector	9
Raw Block Volume support	9
Volume snapshot	10
Volume Cloning	10
Storage Classes	10
Provisioner	11
Parameters	11
Reclaim Policy	11
Mount Options	11
Volume Binding Mode	11
Allowed topologies	11

Volumes

At its core, a volume is just a directory, possibly with some data in it, which is accessible to the Containers in a Pod. How that directory comes to be, the medium that backs it, and the contents of it are determined by the particular volume type used.

To use a volume, a Pod specifies what volumes to provide for the Pod (the `.spec.volumes` field) and where to mount those into Containers (the `.spec.containers.volumeMounts` field).

Volume Types:

- hostPath
- emptyDir
- configMap
- secret
- gcePersistentDisk
- awsElasticBlockStore
- azureDisk
- vSphereVolume
-
- iscsi
- nfs
- scaleIO
- fc (fibre Channel)

The storage media (Disk, SSD, etc.) of an emptyDir volume is determined by the medium of the filesystem holding the kubelet root dir (typically /var/lib/kubelet). There is no limit on how much space an emptyDir or hostPath volume can consume, and no isolation between Containers or between Pods.

Persistent Volume

A PersistentVolume (PV) is a piece of storage in the cluster that has been provisioned by an administrator or dynamically provisioned using [Storage Classes](#).

It is a resource in the cluster just like a node is a cluster resource. PVs are volume plugins like Volumes, but have a lifecycle independent of any individual pod that uses the PV.

A PersistentVolumeClaim (PVC) is a request for storage by a user.

Provisioning

- Static

- A cluster administrator creates a number of PVs.
- They carry the details of the real storage which is available for use by cluster users.
- Example

apiVersion: v1

kind: PersistentVolume

metadata:

name: mongodb-pv

spec:

capacity:

storage: 1Gi

accessModes:

- ReadWriteOnce

- ReadOnlyMany

persistentVolumeReclaimPolicy: Retain

gcePersistentDisk:

pdName: mongodb

fsType: ext4

- Dynamic

- When none of the static PVs the administrator created matches a user's PersistentVolumeClaim, the cluster may try to dynamically provision a volume specially for the PVC.
- This provisioning is based on StorageClasses: the PVC must request a [storage class](#) and the administrator must have created and configured that class in order for dynamic provisioning to occur.

Binding

- A PVC to PV binding is a one-to-one mapping.
- Claims will remain unbound indefinitely if a matching volume does not exist. Claims will be bound as matching volumes become available. For example, a cluster provisioned with many 50Gi PVs would not match a PVC requesting 100Gi.
- The PVC can be bound when a 100Gi PV is added to the cluster

Storage object in use protection

- If a user deletes a PVC in active use by a pod, the PVC is not removed immediately. PVC removal is postponed until the PVC is no longer actively used by any pods, and also if admin deletes a PV that is bound to a PVC, the PV is not removed immediately. PV removal is postponed until the PV is no longer bound to a PVC.

Reclaim

- Retain
 - The Retain reclaim policy allows for manual reclamation of the resource
 - Pod and Claim object is deleted
 - not yet available for another claim because the previous claimant's data remains on the volume
 - PV remains, can be manually deleted by admin and data on the disk can be deleted
 - Disk can be reused by creating another PV
 - Pod deleted
 - PVC/PV/disk do not get deleted
 - Same PVC CANNOT be used to mount to another POD
 - **PVC deleted**
 - **PV remains**
 - **PV deleted**
 - **Actual disk do not get deleted**
 - **Actual disk data can be deleted and made available to another PV**
- Delete
 - Pod using PVC deleted
 - PVC and PV/disk do not get deleted
 - Same PVC can be used to mount to another POD
 - PVC deleted

- PV and disk is deleted
- Recycle
 - Pod using PVC deleted
 - PVC and PV/disk do not get deleted
 - Same PVC can be used to mount to another POD
 - PVC deleted
 - PV and disk is not deleted.
 - Only data is deleted.. **BUT recycle plugin is needed..**

Otherwise recycle operations fails and PV status goes to FAILED state and cannot be claimed.. In GCP by default there is no recycle plugin.. Only Delete and Retain is supported
By default default storage class reclaim type is "DELETE"

```
imsrv01@cloudshell:~ (my-kubernetes-codelab-227201)$ k get pv
NAME          CAPACITY  ACCESS MODES  RECLAIM POLICY  STATUS  CLAIM          STORAGECLASS  REASON  AGE
gcp-pv-volume 1Gi       RWO           Recycle         Failed  default/gcp-pv-claim  local                    11m
```

```
persistentVolumeReclaimPolicy: Recycle
storageClassName: local
status:
  message: No recycler plugin found for the volume!
  phase: Failed
```

- performs a basic scrub (rm -rf /thevolume/*) on the volume and makes it available again for a new claim.

Expanding PVC

You can only expand a PVC if its **storage class's allowVolumeExpansion field is set to true.**

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: gluster-vol-default
provisioner: kubernetes.io/glusterfs
```

```
parameters:  
  resturl: "http://192.168.10.100:8080"  
  restuser: ""  
  secretNamespace: ""  
  secretName: ""  
  allowVolumeExpansion: true
```

To request a larger volume for a PVC, edit the PVC object and specify a larger size. This triggers expansion of the volume that backs the underlying PersistentVolume. A new PersistentVolume is never created to satisfy the claim. Instead, an existing volume is resized.

Persistent Volume Spec

```
apiVersion: v1  
kind: PersistentVolume  
metadata:  
  name: pv0003  
spec:  
  capacity:  
    storage: 5Gi  
  volumeMode: Filesystem  
  accessModes:  
    - ReadWriteOnce  
  persistentVolumeReclaimPolicy: Recycle  
  storageClassName: slow  
  mountOptions:  
    - hard  
    - nfsvers=4.1
```

nfs:

path: /tmp

server: 172.17.0.2

Capacity

VolumeMode (optional)

- create a filesystem on the persistent volume or just a raw device ?
- filesystem is the default
- Other - block

Access Mode

The access modes are:

- ReadWriteOnce – the volume can be mounted as read-write by a single node
- ReadOnlyMany – the volume can be mounted read-only by many nodes
- ReadWriteMany – the volume can be mounted as read-write by many nodes

In the CLI, the access modes are abbreviated to:

- RWO - ReadWriteOnce
- ROX - ReadOnlyMany
- RWX - ReadWriteMany

Class

A PV can have a class, which is specified by setting the `storageClassName` attribute to the name of a [StorageClass](#). A PV of a particular class can only be bound to PVCs requesting that class. **A PV with no `storageClassName` has no class and can only be bound to PVCs that request no particular class.**

Reclaim Policy

Phase

A volume will be in one of the following phases:

- **Available – a free resource that is not yet bound to a claim**

- **Bound** – the volume is bound to a claim
- **Released** – the claim has been deleted, but the resource is not yet reclaimed by the cluster
- **Failed** – the volume has failed its automatic reclamation

Node Affinity

1. A PV can specify [node affinity](#) to define constraints that limit what nodes this volume can be accessed from. Pods that use a PV will only be scheduled to nodes that are selected by the node affinity
2. Mainly for local volumes

Persistent Volume Claim Spec

```

apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: myclaim
spec:
  accessModes:
    - ReadWriteOnce
  volumeMode: Filesystem
  resources:
    requests:
      storage: 8Gi
  storageClassName: slow
  selector:
    matchLabels:
      release: "stable"
  matchExpressions:
    - {key: environment, operator: In, values: [dev]}
```

Access Mode

- Same as PV

VolumeMode

- Same as PV

Resources

- Same as Pods

Class

- A claim can request a particular class by specifying the name of a [StorageClass](#) using the attribute `storageClassName`. Only PVs of the requested class, ones with the same `storageClassName` as the PVC, can be bound to the PVC.
- PVCs don't necessarily have to request a class. A PVC with its `storageClassName` set equal to "" is always interpreted to be requesting a PV with no class, so it can only be bound to PVs with no class (no annotation or one set equal to "")
- When a PVC specifies a [selector](#) in addition to requesting a [StorageClass](#), the requirements are ANDed together: only a PV of the requested class and with the requested labels may be bound to the PVC.
- Default storage class - Specifying a default StorageClass is done by setting the annotation `storageclass.kubernetes.io/is-default-class` equal to "true" in a StorageClass object.
- Admission plugin - turned ON and OFF -- try examples..

Selector

Claims can specify a [label selector](#) to further filter the set of volumes. Only the volumes whose labels match the selector can be bound to the claim. The selector can consist of two fields:

- `matchLabels` - the volume must have a label with this value
- `matchExpressions` - a list of requirements made by specifying key, list of values, and operator that relates the key and values. Valid operators include `In`, `NotIn`, `Exists`, and `DoesNotExist`.

All of the requirements, from both `matchLabels` and `matchExpressions` are ANDed together – they must all be satisfied in order to match.

Raw Block Volume support

`volumeMode: Block`

Try examples...

Volume snapshot

Volume Cloning

Storage Classes

- A StorageClass provides a way for administrators to describe the “classes” of storage they offer. Different classes might map to quality-of-service levels, or to backup policies, or to arbitrary policies determined by the cluster administrators. Kubernetes itself is unopinionated about what classes represent
- Each StorageClass contains the fields provisioner, parameters, and reclaimPolicy, which are used when a PersistentVolume belonging to the class needs to be dynamically provisioned.

apiVersion: storage.k8s.io/v1

kind: StorageClass

metadata:

name: standard

provisioner: kubernetes.io/aws-ebs

parameters:

type: gp2

reclaimPolicy: Retain

mountOptions:

- debug

volumeBindingMode: Immediate

Provisioner

Parameters

Reclaim Policy

Mount Options

Volume Binding Mode

Immediate mode -

volume binding and dynamic provisioning occurs once the PersistentVolumeClaim is created. For storage backends that are topology-constrained and not globally accessible from all Nodes in the cluster, PersistentVolumes will be bound or provisioned without knowledge of the Pod's scheduling requirements. This may result in unschedulable Pods

WaitForFirstConsumer mode which will delay the binding and provisioning of a PersistentVolume until a Pod using the PersistentVolumeClaim is created. PersistentVolumes will be selected or provisioned conforming to the topology that is specified by the Pod's scheduling constraints

Allowed topologies

apiVersion: storage.k8s.io/v1

kind: StorageClass

metadata:

name: standard

provisioner: kubernetes.io/gce-pd

parameters:

type: pd-standard

volumeBindingMode: WaitForFirstConsumer

allowedTopologies:

- matchLabelExpressions:






- key: failure-domain.beta.kubernetes.io/zone

values:

- us-central1-a

- us-central1-b

Cloud Storage

	 Azure	 Google	 AWS
Object Storage	Azure Blob Storage	Google Cloud Storage	Amazon Simple Storage Service (S3)
Virtual Machine / Block Storage	Azure Page Blobs / Premium Storage	Persistent Disk	Amazon Elastic Block Storage (EBS)
File Storage	Azure File Storage		Amazon Elastic File System (EFS)
Long Term Cold Storage	Azure Cool Storage	Google Coldline Storage	Amazon Glacier
Hybrid / Gateway Storage	Azure StorSimple		AWS Storage Gateway

What is Cloud Native?

Horizontally scalable

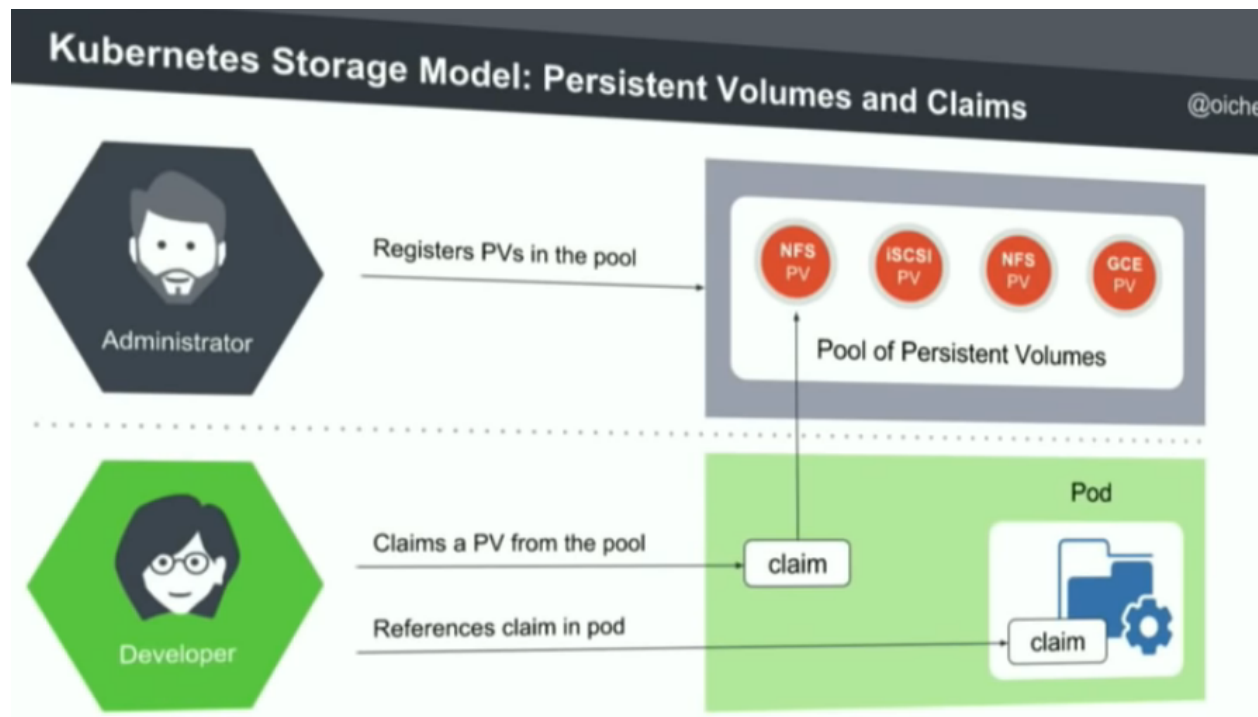
No single point of failure

Resilient and self healing

Minimal operator overhead

Decoupled from the underlying platform

Problem - Storage/volume needs to be created manually..

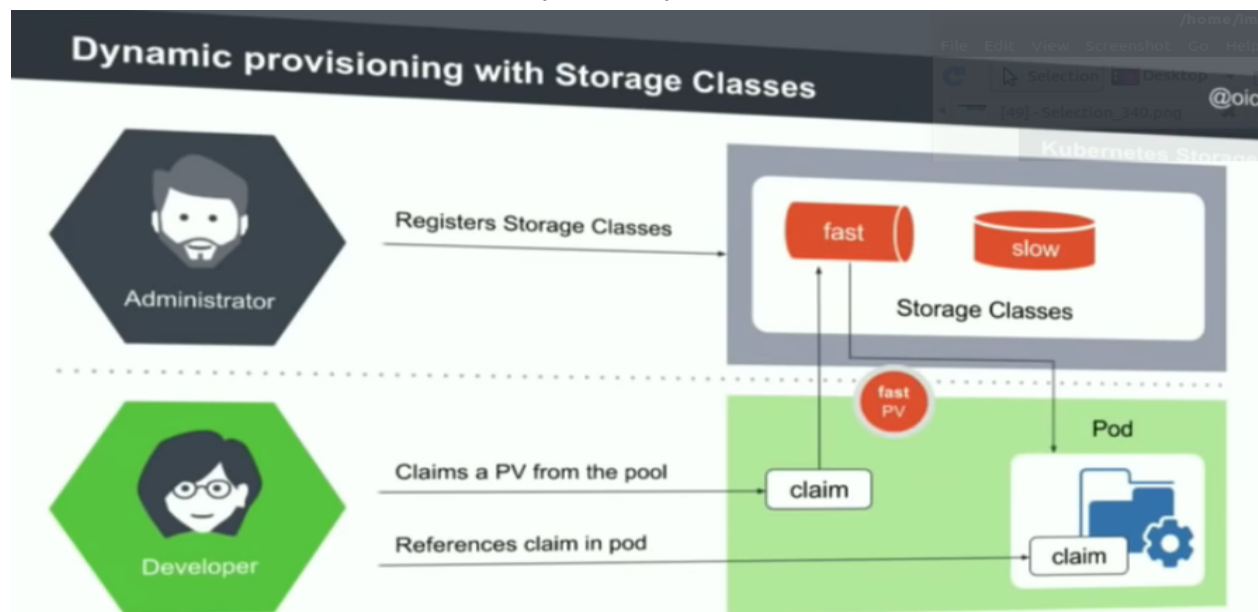


Admin - defines the storage class -- like storage tier.. (Economy/Premium)

Fast storage -- > provision on AWS EBS..

Slow storage -- > google on XXX

When claim request comes , volume is dynamically created.....



Key information

1. What is my **use case**?
2. What are my **performance requirements**?
3. How should developers **access** storage?
4. Where is the storage **deployed and managed**?

2. What are my performance requirements?

@oichery



App Binaries
Ephemeral



App data
Latency,
availability,
performant



Config
Shared



Backup
Cost efficient,
cloud

3. How should developers access storage?

@oiche



Block

Fixed-size 'blocks' in a rigid arrangement – ideal for enterprise databases



File

'Files' in hierarchically nested 'folders' – ideal for active documents

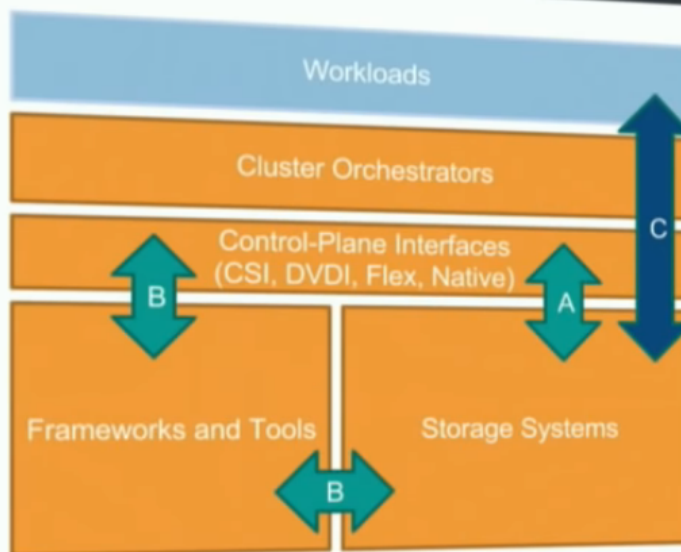


Object

'Objects' in scalable 'buckets' – ideal for unstructured big data and archiving

4. Where is the storage deployed and managed?

@oiche



- CO supports one or more **Interfaces** to interact with the Storage System

- Storage System can **(A)** support control-plane interface API directly and interact directly with the CO or can **(B)** interact with the CO via an **API framework layer** or other **Tools**.

- Storage system must support the ability to provision and consume **(C)** volumes through a standard interface to be considered **Interoperable**

- Workloads interact **(C)** with storage systems over various data-plane methods

Jane's storage requirements

@oich



- Database location, credentials
- Postgres database for application data
- User uploaded media
- Database and website backups

Database location and credentials

@oichery

1. **Use case?** Configuration
2. **Performance requirements?** Shared across instances
3. **Access?** Kubernetes provides Secrets for sensitive data such as passwords, and ConfigMap for arbitrary config. Both can be accessed by the application through environment variables
4. **Deployed and managed?** Tight integration with Kubernetes

User uploaded media

@oi

1. **Use case?** Shared media
2. **Performance requirements?** Large blobs of data, shared across pods
3. **Access?** Shared filesystem
4. **Deployed and managed?**

Cloud: Managed NFS, or object store if the app can support it

On prem: Distributed FS (but please not NFS!)

Database and website backup

1. **Use case?** Backup and archival
2. **Performance requirements?** Durability, cost, snapshots
3. **Access?** Object store
4. **Deployed and managed?**

Cloud: Managed object store, long term cold storage

On prem: Object store, NAS

1. **Use case?** Transactional database
2. **Performance requirements?** High availability, low latency, deterministic performance
3. **Access?** Database connector
4. **Deployed and managed?**

Cloud: Cloud volumes (watch out for attach/detach times, compliance) or managed db (limited offerings)

On prem: Software defined storage