

TECHNOLOGY



Certified Kubernetes Administrator

Storage



Learning Objectives

By the end of this lesson, you will be able to:

- 🕒 Discuss Storage in Kubernetes
- 🕒 State the different types of Volumes
- 🕒 Present an overview of Storage Classes
- 🕒 Discuss Dynamic Volume Provisioning
- 🕒 List the different types of Ephemeral Volumes



Overview of Storage in Kubernetes

Persistent Storage

Kubernetes gained importance as a method for hosting microservice-based processes and data storage owing to its Persistent Storage.

Users can create databases and access them too. This can also be done for data for various applications.

By using StorageClass, administrators can assign “classes” of various storage-to-map service quality levels.

Users can add arbitrary policies and backup policies assigned by Cluster administrators.



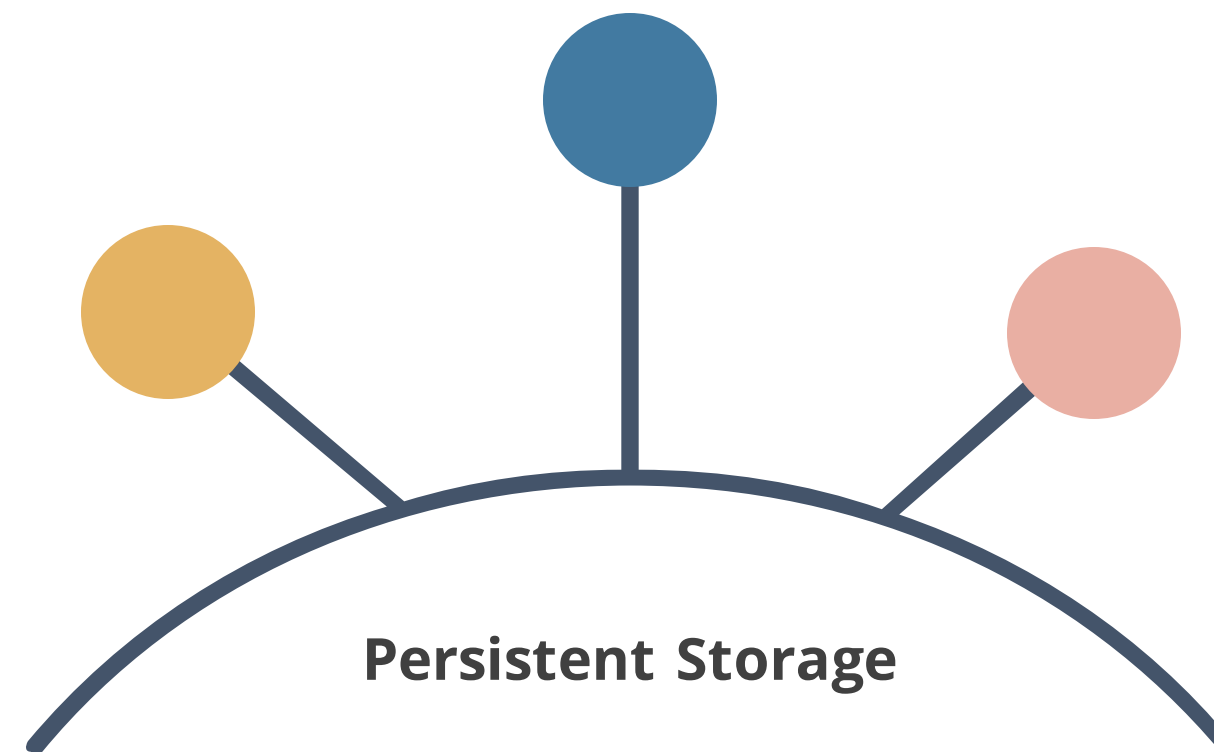
Kubernetes and Persistent Storage

Kubernetes helps in Persistent Storage as it does not have any association with volatile Containers.

Information cannot be shared among applications of a wheelhouse if there is deletion of localized data.

Applications need to remain functional and be in their unaltered state.

Kubernetes helps to store data outside the Container, where it can be assessed without interruption.



Requirements for Persistent Storage



Backend

Kubernetes Persistent Storage hides the data from applications and the users. It includes protocols such as NFS, iSCSI, and SMB.



Storage services and systems on cloud providers give more people access to user information.



Third-party cloud storage builds environments where users have full access to data, which they can integrate.

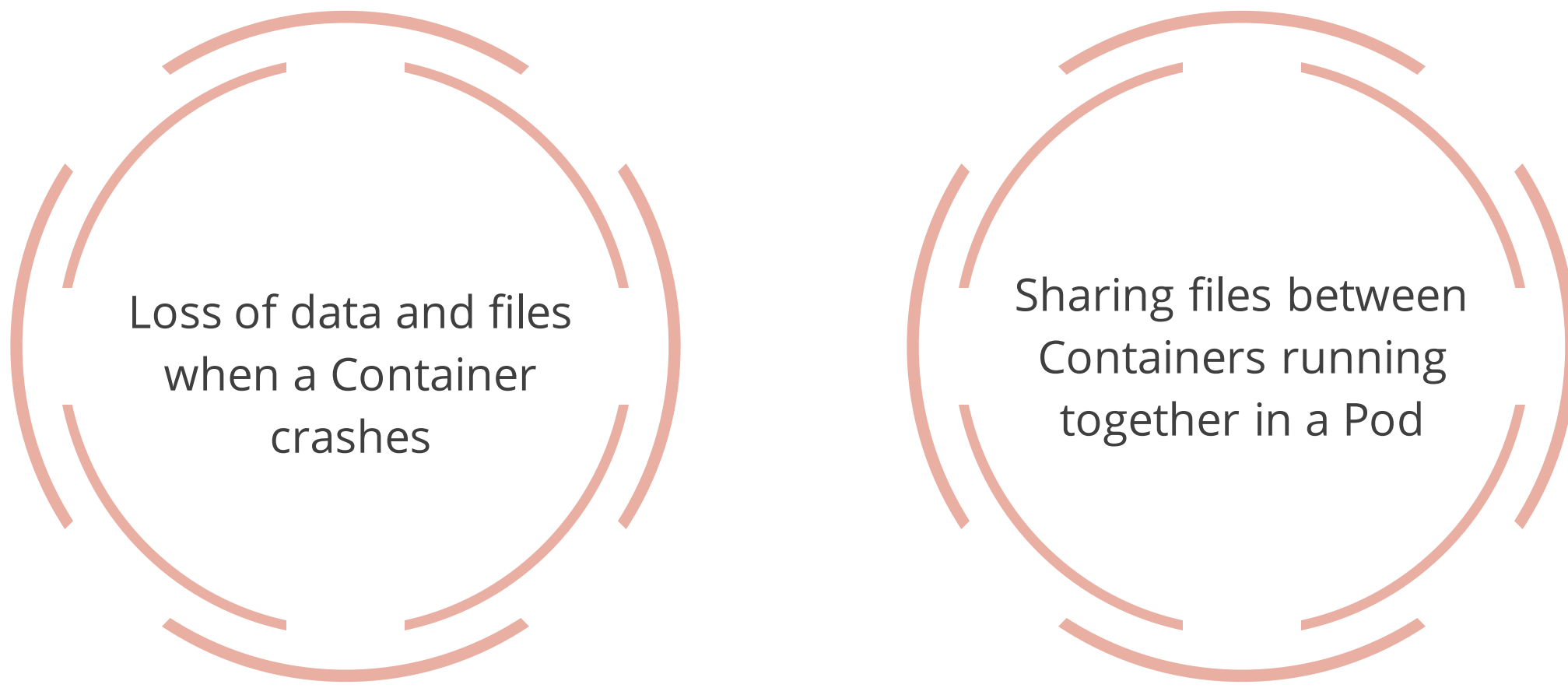


Storage providers such as Amazon S3 or LINBIT offer a selection of tools and applications, and help in Persistent Storage.

Volumes

Overview

Kubernetes Volume Abstraction helps business-critical applications running in Containers solve problems.



Loss of data and files
when a Container
crashes

Sharing files between
Containers running
together in a Pod

Background

A Pod can have several simultaneous Volumes and Volume types while using Kubernetes. Volumes cannot have higher positions over other Volumes or be connected with hard links to other Volumes.



A Volume's lifespan is greater than any Container running within the Pod. Data preservation is done across the Container.



For a Volume's use, it should be specified to provide for the Pod in **.spec.volumes** and it should declare the Containers for mounting those Volumes in **.spec.containers[*].volumeMounts**.

Types of Volumes

Kubernetes supports many types of Volumes:

awsElasticBlockStorage

azureDisk

azureFile

cephfs

cinder

configMap

downwardAPI

emptyDir

fibre channel

gcePersistentDisk

glusterfs

hostPath

Types of Volumes

iscsi

local

nfs

persistentVolumeClaim

portworxVolume

projected

quobyte

rbd

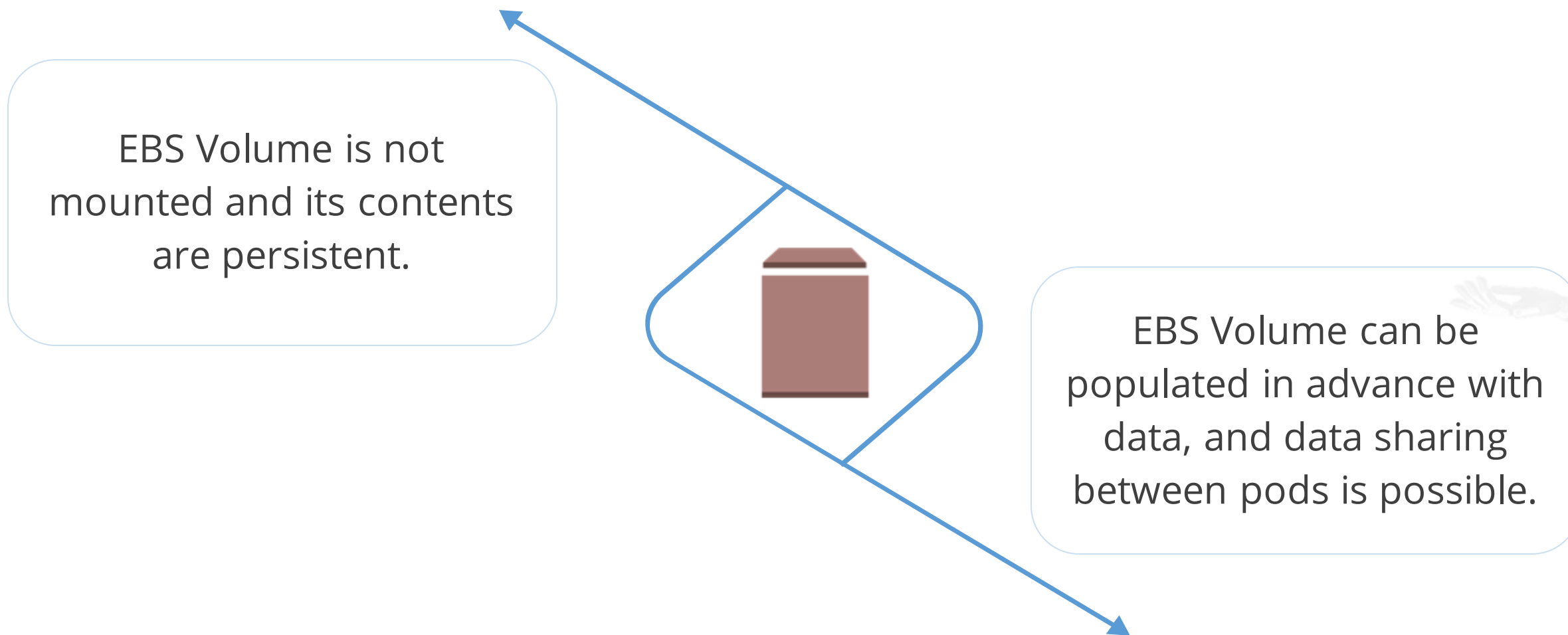
scaleIO (deprecated)

storageOS

vsphereVolume

awsElasticBlockStore

awsElasticBlockStore Volume is a part of the Amazon Web Services (AWS) EBS Volume.



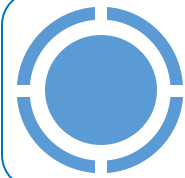
azureDisk

Microsoft's Azure Data Disk gets mounted onto a pod with azureDisk Volume type.

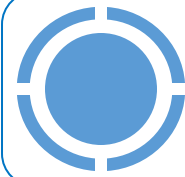


cephfs

A cephfs Volume allows the mounting of an existing cephfs Volume to your Pod.



A cephfs Volume is unmounted and its contents are preserved.



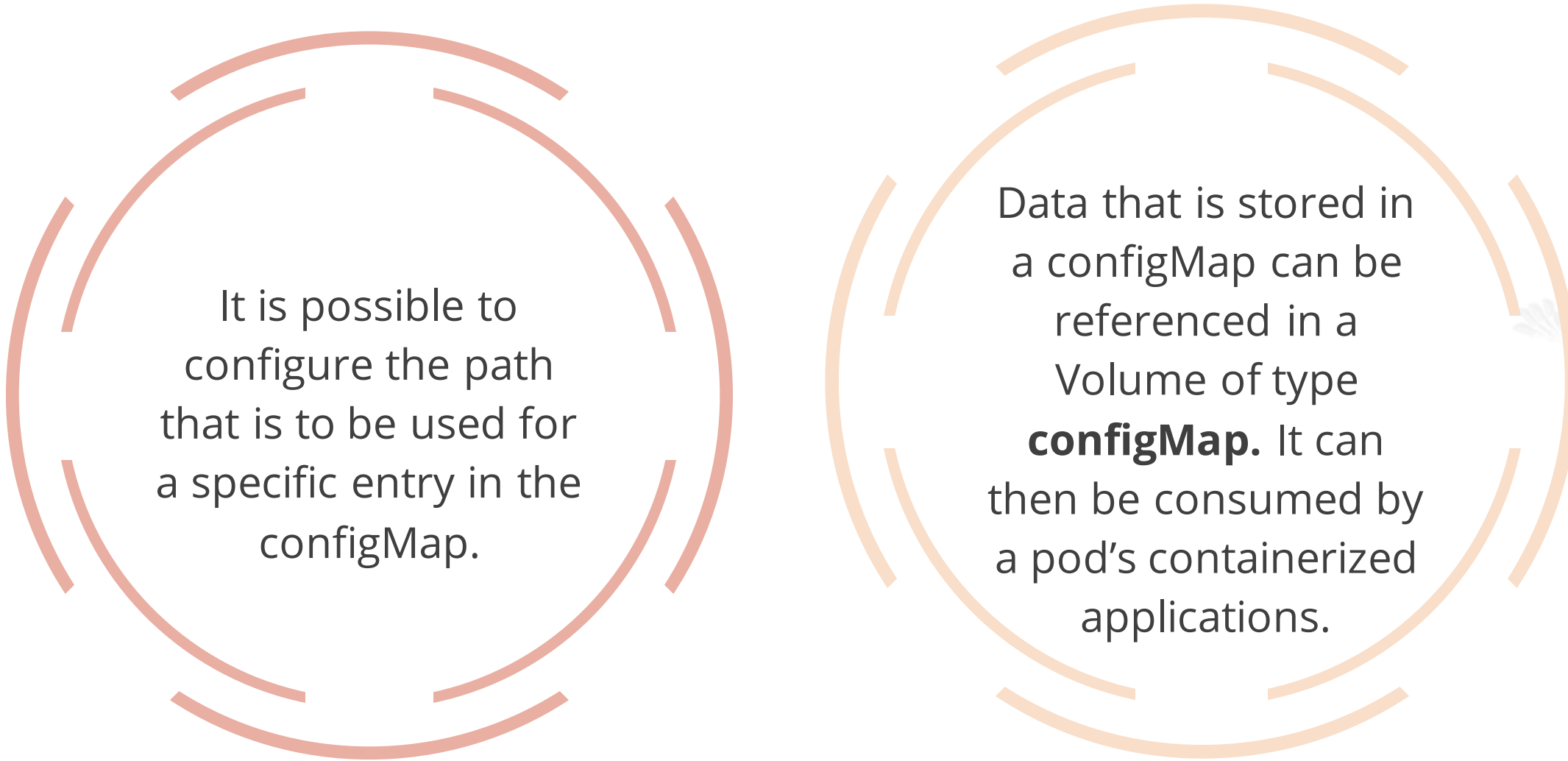
cephfs Volume can be prepopulated with data.



Multiple writers can simultaneously mount onto a cephfs Volume.

configMap

ConfigMaps are used to inject configuration data into Pods.



It is possible to configure the path that is to be used for a specific entry in the configMap.

Data that is stored in a configMap can be referenced in a Volume of type **configMap**. It can then be consumed by a pod's containerized applications.

emptyDir

An emptyDir volume gets made when a Pod is assigned to a Node. It exists for the duration that the Pod runs on that Node.

Uses of emptyDir:

- ✓ Scratch space, e.g., a disk-based merge sort
- ✓ Checkpointing a long computation for recovery from crashes
- ✓ Holding files fetched by a content-manager Container while data is served by a web server Container

gcePersistentDisk and iscsi

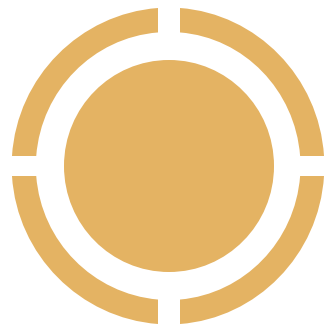
A **gcePersistentDisk** Volume is used to mount a Google Compute Engine (GCE) Persistent Disk (PD) onto the Pod.

Persistent disks can be populated in advance with data. It is possible to share this data between Pods.

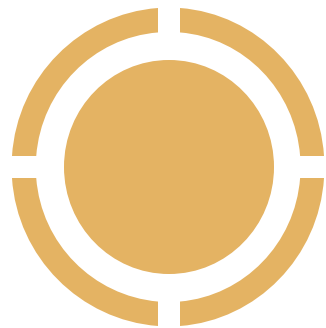
An **iscsi** Volume allows an existing iSCSI (SCSI over IP) Volume to be mounted onto the Pod.

Iscsi can be mounted as read-only simultaneously by multiple consumers.

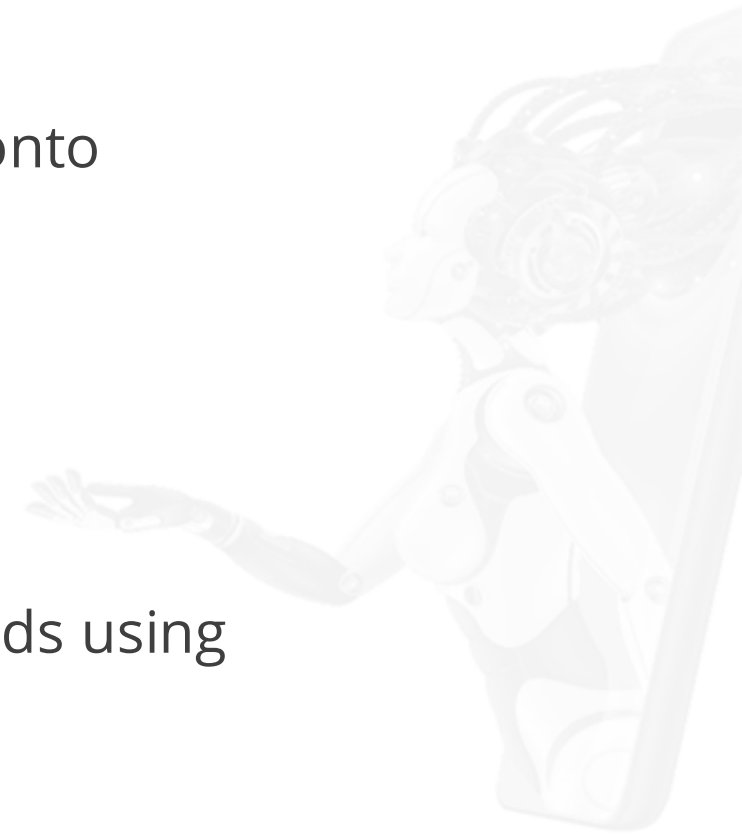
nfs and secret



An existing NFS (Network File System) share can be mounted onto a Pod using an nfs Volume.



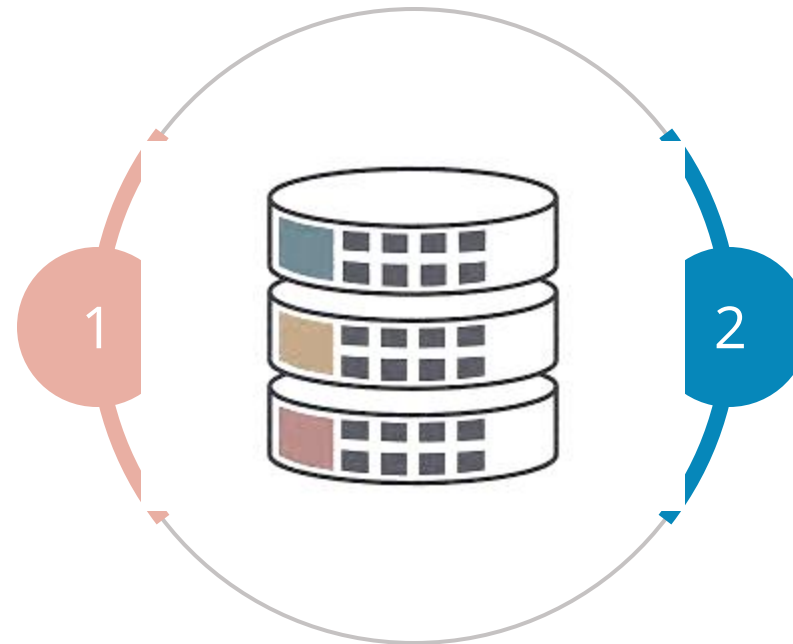
Sensitive information, such as passwords, can be passed to Pods using a secret Volume.



storageOS

An existing storageOS volume can be mounted onto a Pod using a storageOS volume. StorageOS runs as a Container with Kubernetes environment, and results in local or attached storage being accessible from any Node within the Kubernetes Cluster.

StorageOS can be used to replicate data and create protection against node failure.



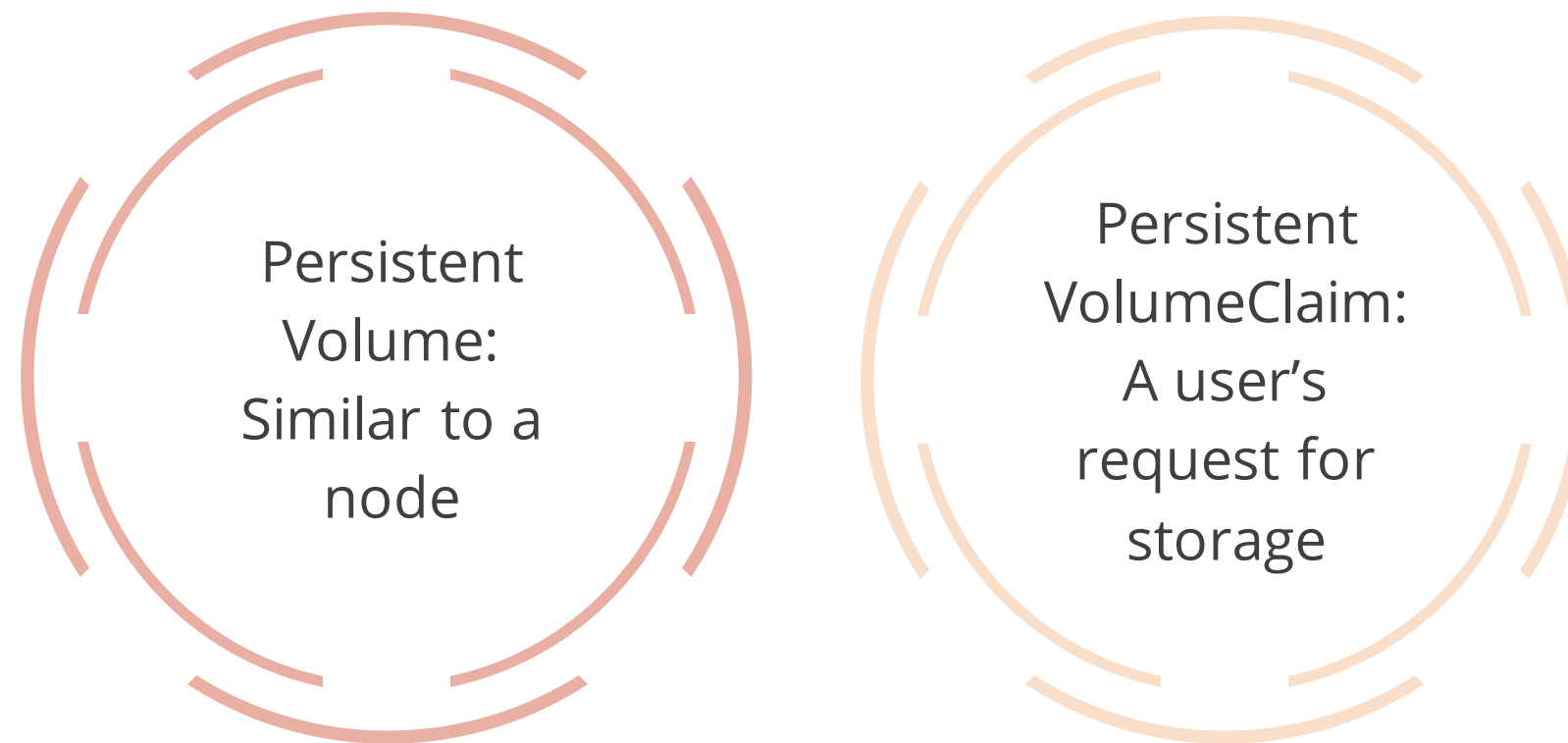
StorageOS provides block storage to Containers accessible from a file system.

Persistent Volumes

Introduction

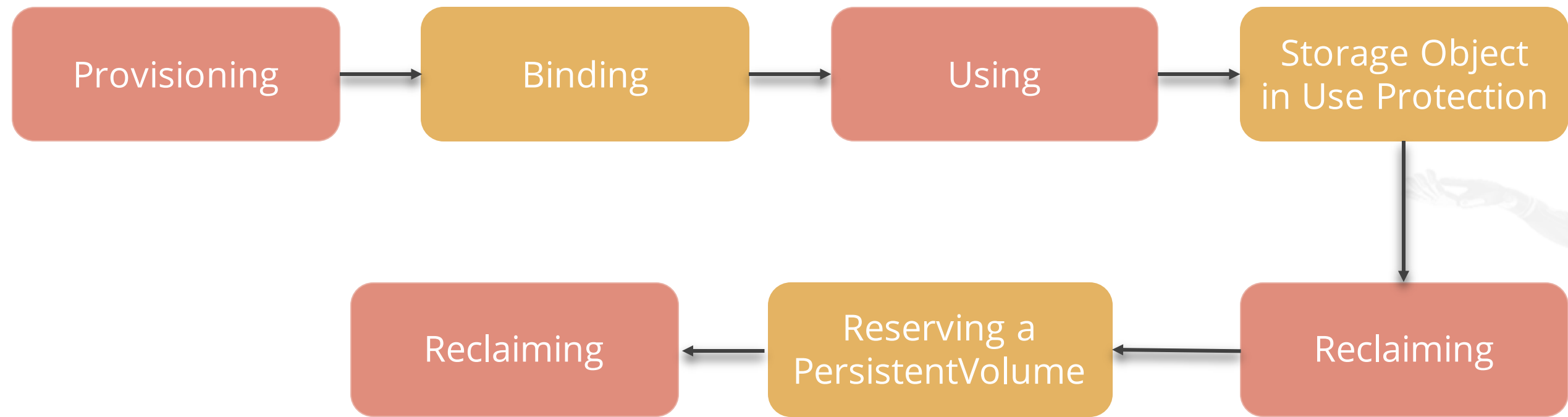
Users and administrators get an API from the PersistentVolume sub-system in the Kubernetes orchestration system. This API abstracts storage related details.

Kubernetes technology introduces two new API resources:



Life cycle of Volume and Claim

The life cycle of interaction between PVs and PVCs:



Types of Persistent Volumes

Some of the non-deprecated plugins that Kubernetes currently supports are:

1

awsElasticBlockStore - AWS Elastic Block Store (EBS)

2

azureDisk and azureFile - Azure Disk and Azure File from Microsoft

3

cephfs - CephFS Volume

4

csi - Container Storage Interface (CSI)

5

fc - Fibre Channel (FC) Storage

6

flexVolume - FlexVolume

7

flocker - Flocker Storage

8

gcePersistentDisk - GCE Persistent Disk

Types of Persistent Volumes

9

glusterfs - Glusterfs Volume

10

scsi - iSCSI (SCSI over IP) Storage

11

local - local storage devices mounted on Nodes

12

nfs - Network File System (NFS) Storage

13

portworxVolume - Portworx Volume

14

quobyte - Quobyte Volume

15

rbd - Rados Block Device (RBD) Volume

16

storageos - StorageOS Volume

Spec for Persistent Volumes

Capacity

Volume Mode

Access Mode

Class

Reclaim Policy

Mount Options

Node Affinity

Phase

Persistent Volumes

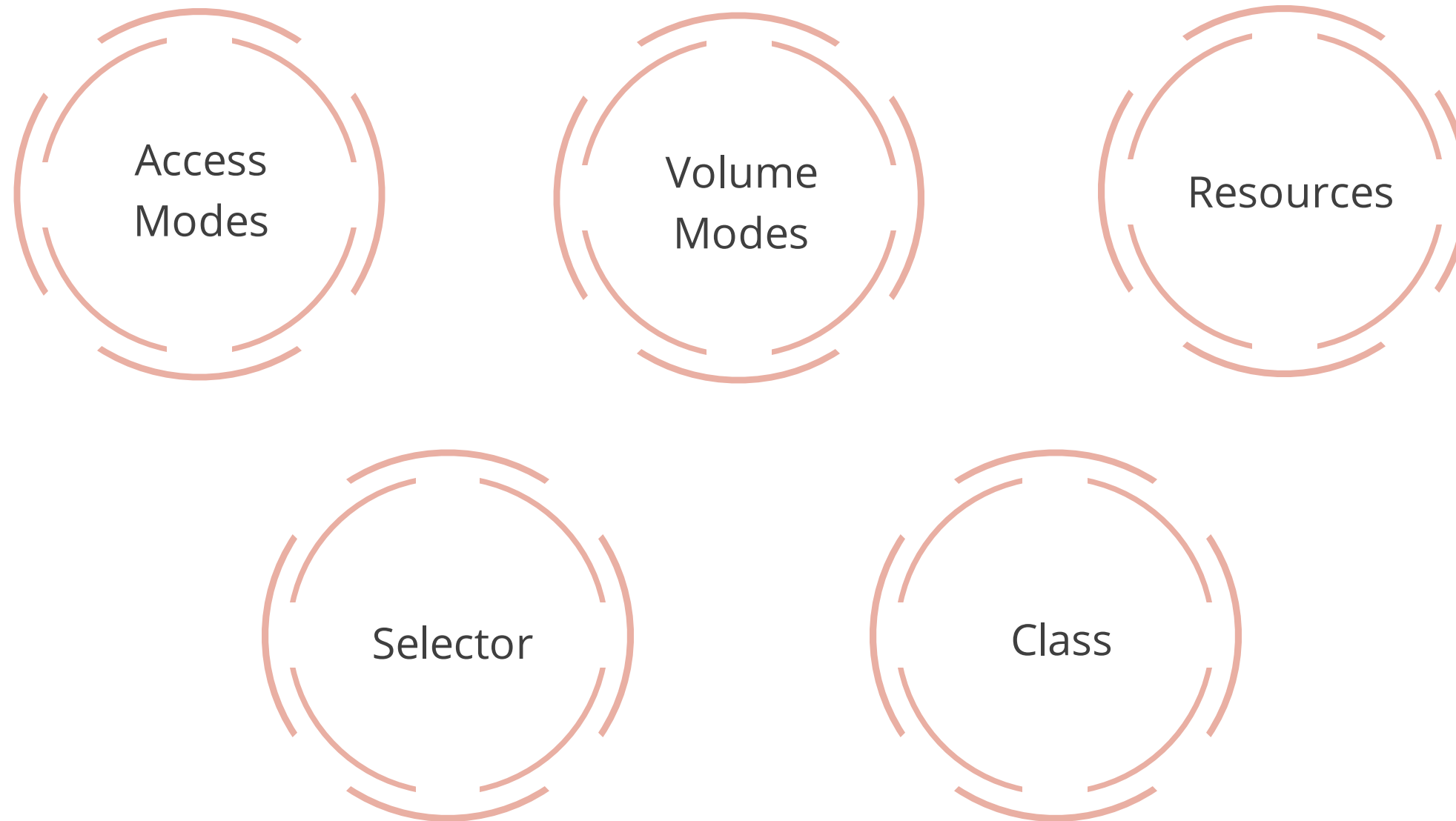
Each PersistentVolume (PV) contains a spec and status — the Volume's specification and status.

Demo

```
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
```



Spec for PersistentVolumeClaim



PersistentVolumeClaim

Each PersistentVolumeClaim (PVC) contains a spec and status — the Claim's specification and status.

Demo

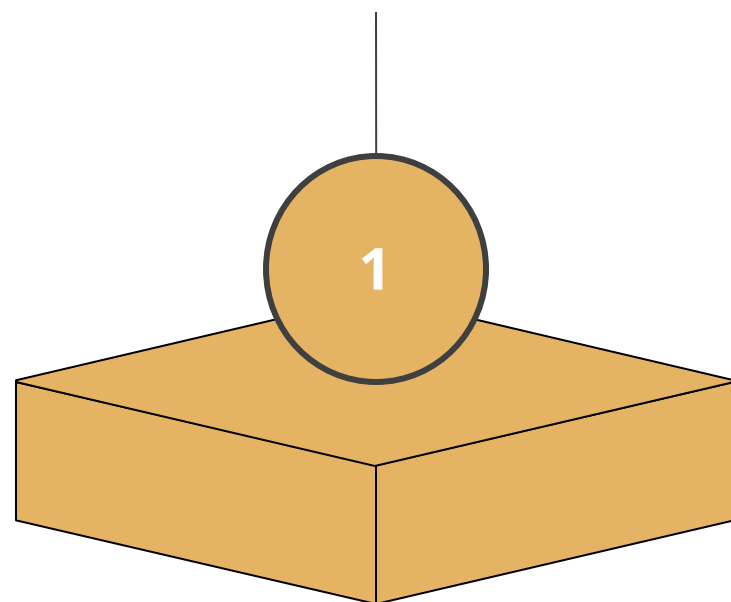
```
apiVersion: v1
Kind: PersistentVolumeClaim
Metadata:
  name: myclaim
spec:
  accessModes:
    - ReadWriteOnce
  volumeMode: Fsystem
  resources:
    requests:
      storage: Bgi
  storageClassName : slow
  selector:
    matchLabels:
      release: "stable"
    matchExpressions:
      - {key:
environment, operator : In, values:[dev]}
```



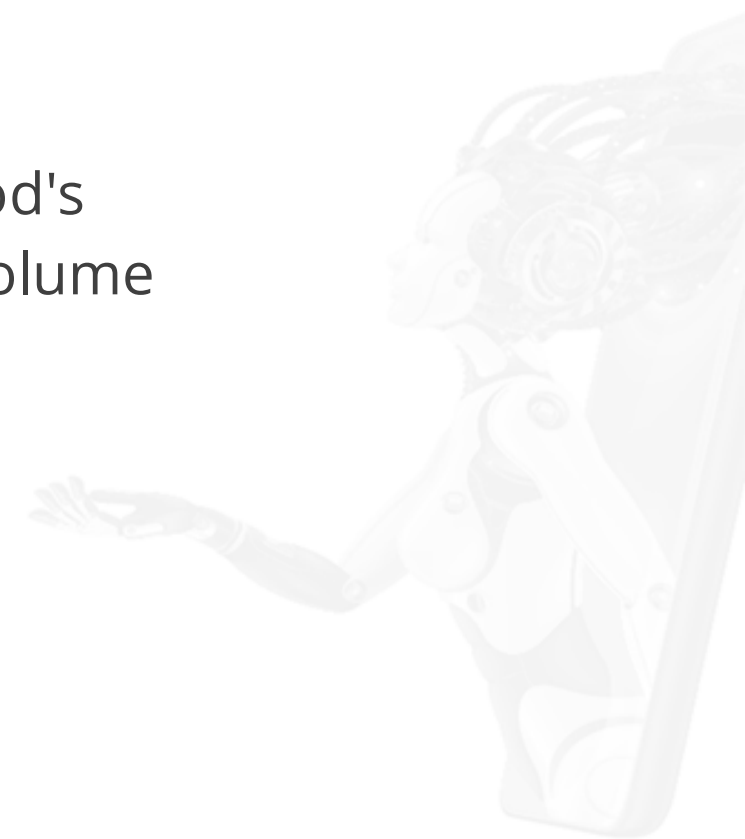
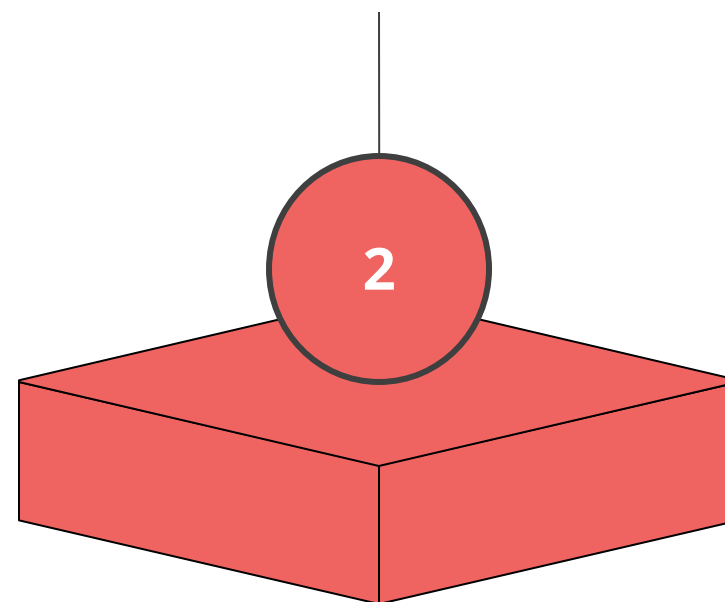
Claim as Volumes

Pods access storage by utilizing Claims as Volumes.

Claims must remain in the same namespace as the Pod that uses the Claim.



Clusters use the Claim in the Pod's namespace to get the PersistentVolume that backs the Claim.



Raw Block Volume Support

Raw block materials are supported by these Volume plugins:

AWSElasticBlockStore

AzureDisk

CSI

FC (Fibre Channel)

GCEPersistentDisk

iSCSI

Local volume

OpenStack Cinder

RBD (Ceph Block
Device)

VsphereVolume

PersistentVolume Using a Raw Block Volume

Here is how to configure a PersistentVolume that uses a Raw Block Volume for an accessMode of ReadWriteOnce:

Demo

```
apiVersion: v1
Kind: PersistentVolume
Metadata:
  name: block-pv
spec:
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  volumeMode: Block
  persistentVolumeClaimPolicy: Retain
  fc:
    targettwWWNs: { "50060e801049cfd1" }
    lun: 0
    readOnly: false
```



PersistentVolumeClaim Requesting a Raw Block Volume

Given below is how a PersistentVolumeClaim that requests for a Raw Block Volume for an accessMode of ReadWriteOnce may be configured:

Demo

```
apiVersion: v1
Kind: PersistentVolumeClaim
Metadata:
  name: block-pvc
Spec:
  accessModes:
    - ReadWriteOnce
  volumeMode: Block
  resources:
    requests:
      storage: 10Gi
```



Pod Specification Adding Raw Block Device Path in Container

Demo

```
apiVersion: v1
Kind: Pod
Metadata:
  name :    pod-with-block-volume
Spec:
  Containers:
    - name:    fc-container
      image:    fedora:26
      Command:  ["/bin/sh", "-c"]
      args:     [ " tall -f /dev/null" ]
      volumeDevices:
        - name:    data
          devicePath: /dev/xvda
  volume :
    - name:    data
      persistentVolumeClaim:
        claimName: block-pvc
```



Binding Block Volumes

PV volumeMode	PVC volumeMode	Result
Unspecified	Unspecified	BIND
Unspecified	Block	NO BIND
Unspecified	Filesystem	BIND
Unspecified	Unspecified	NO BIND
Block	Block	BIND
Block	Filesystem	NO BIND
Filesystem	Filesystem	BIND
Filesystem	Block	NO BIND
Filesystem	Unspecified	BIND



Understanding the Working of Persistent Volumes



Problem Statement:

Understand the working of Persistent Volumes in Kubernetes.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to demonstrate Persistent Volumes in Kubernetes:

1. Creating an index.html file on the master node
2. Creating a basic persistent volume
3. Creating a persistent volume claim
4. Creating a pod using persistent volume claim



Volume Snapshots

Introduction

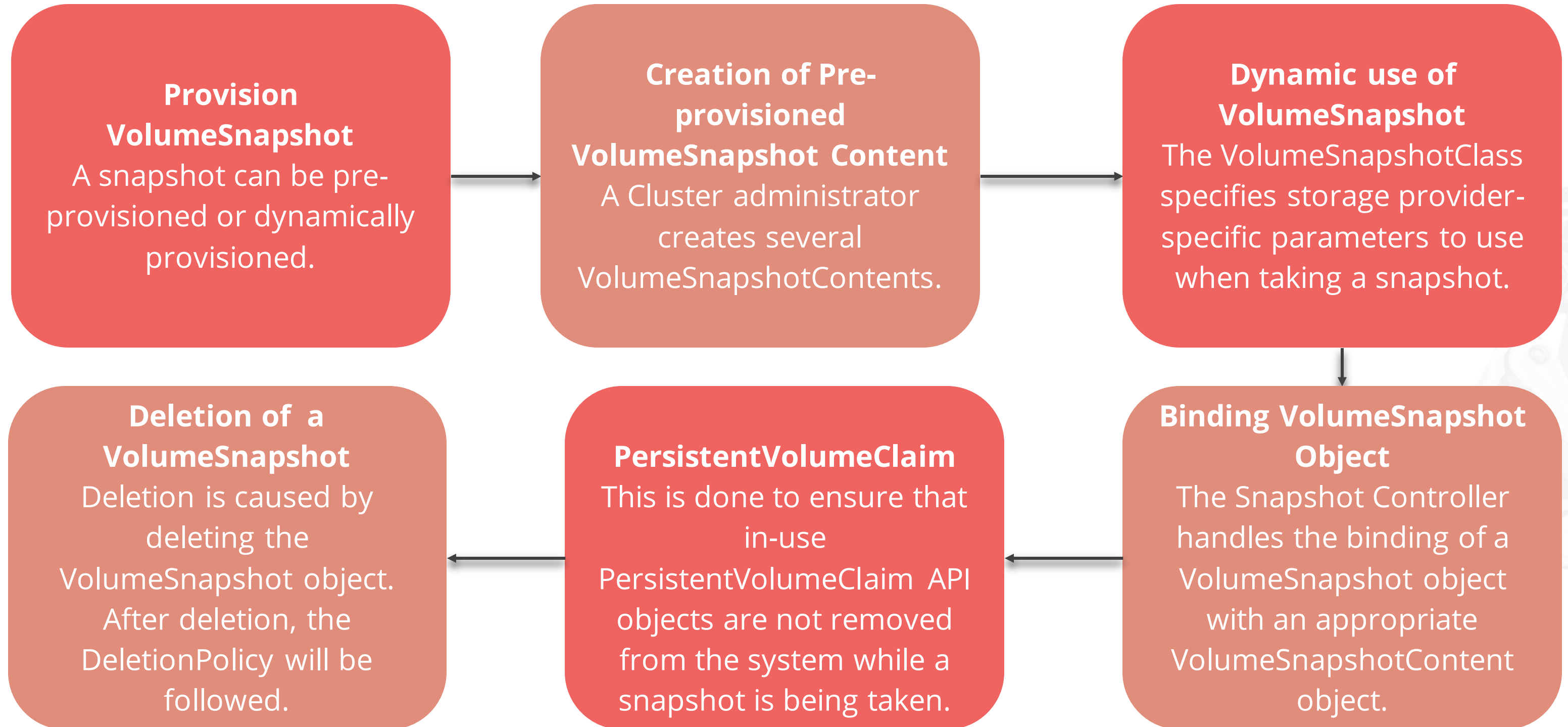
VolumeSnapshotContent and VolumeSnapshot API resources enable the creation of Volume Snapshots for users and administrators.

A VolumeSnapshotContent is a snapshot originating from a Volume in the Cluster that has been provisioned by an administrator.

A VolumeSnapshot is a user's request for a snapshot of a Volume. This snapshot is similar to a PersistentVolumeClaim.

The VolumeSnapshotClass allows the specification of different attributes of a VolumeSnapshot.

Lifecycle of VolumeSnapshot and VolumeSnapshot Content



Volume Snapshots

A spec and a status are parts of a VolumeSnapshot.

Demo

```
apiVersion: snapshot.storage.k8s.io/v1
Kind: VolumeSnapshot-
Metadata:
  name: new-snapshot-test
spec:
  volumeSnapshotClassName: csi-hostpath-snapclass
  source:
    persistentVolumeClaimName: pvc-test
```



Volume Snapshots

The volumeSnapshotContentName source field is required for pre-provisioned snapshots.

Demo

```
apiVersion:  snapshor.storage.k8s.io/v1
Kind:        VolumeSnapshot
Metadata:
  name:      test-snapshot
Spec:
  source:
    volumeSnapshotContentName: test-contest
```



VolumeSnapshot Contents

In Dynamic Provisioning, the Snapshot Common Controller creates a VolumeSnapshotContent object.

Demo

```
apiVersion:  snapshor.storage.k8s.io/v1
Kind:        VolumeSnapshot
Metadata:
  name: snapcontent- 72d9a349-aacd-42d2-a248-d77560d2455
Spec:
  deletionPolicy: Delete
  driver: hostpath.csi.k8s.io
  source:
    volumeHandle: ee0cfb94-f8d4-11e9-b2d8-0242ac1110002
  volumeSnapshotClassName: csi-hostpath-snapclass
  volumeSnapshotRef:
    name: new-snapshot-test
    namespace: default
    vid: 72d9a349-aacd-42d2-a240-d775650d2455
```



VolumeSnapshot Contents

For pre-provisioned snapshots, the Cluster Administrator creates the VolumeSnapshotContent object.

Demo

```
apiVersion:  snapshor.storage.k8s.io/v1
Kind:        VolumeSnapshot
Metadata:
  name:      new-snapshot-content-test
Spec:
  deletionPolicy:  Delete
  driver:          hostpath.csi.k8s.io
  source:
    snapshotHandle:  7bdd0de3-aaeb-9aae-
0242ac110002
    volumeSnapshpotRef:
      name:  new-snapshot-test
      namespace:  default
```



Provisioning Volumes from Snapshots



New volume, populated in advance with data from a snapshot, can be provisioned with a datasource field in the Persistentvolumeclaim.

Understanding the Working of Volume Snapshots



Problem Statement:

Understand the working of Volume Snapshots.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to demonstrate Volume Snapshots in Kubernetes:

1. Clone the VolumeSnapshot from the existing PersistentVolumeClaim
2. Create a VolumeSnapshot configuration file
3. Create the resource
4. View the PersistentVolumeClaim



CSI Volume Cloning

Introduction

The CSI Volume Cloning feature adds support for specifying existing PVCs in the `dataSource` field. This indicates that a user wants to clone a Volume.



A Clone is defined as a duplicate of an existing Kubernetes Volume that can be consumed like any standard Volume.



From the perspective of the Kubernetes API, cloning adds the ability to specify an existing PVC as a `dataSource` during new PVC creation.

Provisioning

Clones are provisioned like any other PVC except that a dataSource that references an existing PVC in the same namespace is added.

Demo

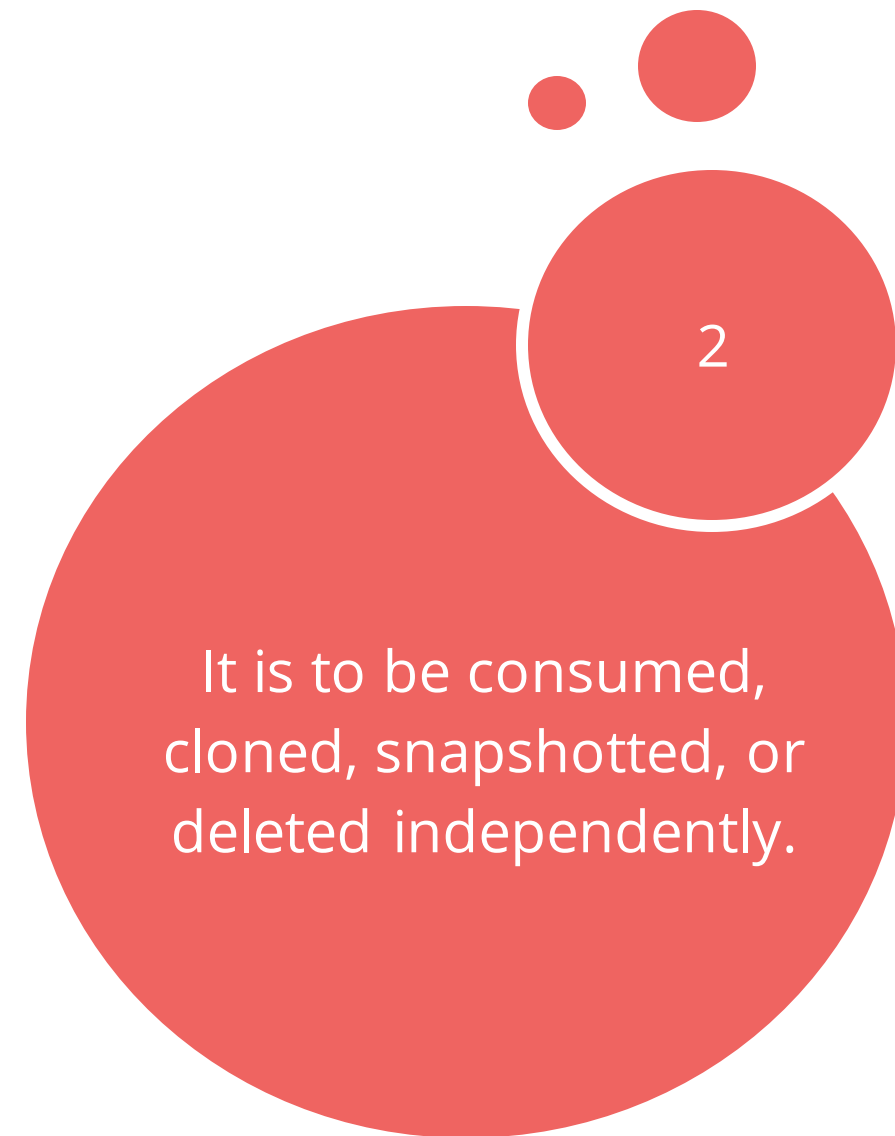
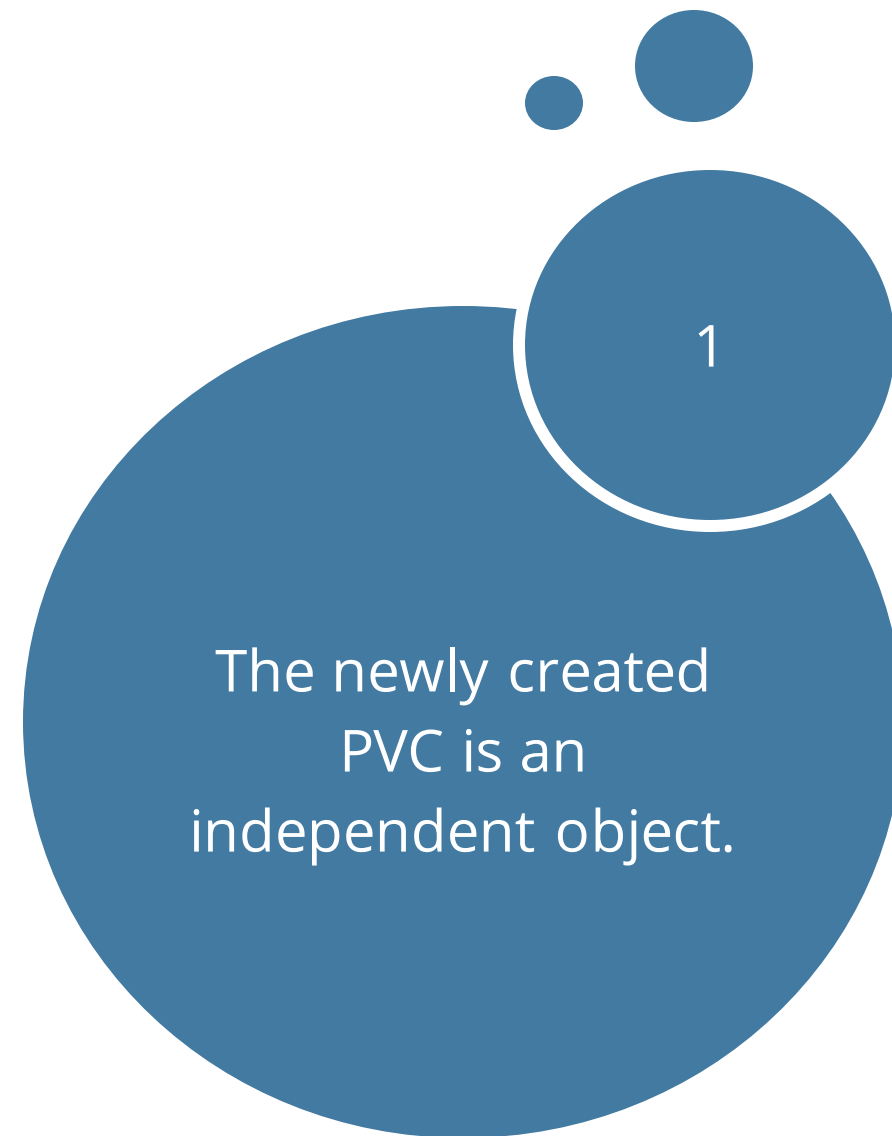
```
apiVersion: v1
Kind: PersistentVolumeClaim
Metadata:
  name: clone-of-pvc-1
  namespace: myns
Spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: cloning

  resources:
    requests:
      storage: 5Gi
  dataSource:
    kind: PersistentVolumeClaim
    name: pvc-1
```



Usage of PVC

When a new PVC is available, the cloned PVC is consumed like any other PVC.



Understanding CSI Volume Cloning



Problem Statement:

Learn how to clone a CSI volume in Kubernetes.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to demonstrate CSI Volume Cloning in Kubernetes:

1. Create a cloned PersistentVolume
2. Create a VolumeSnapshot configuration file
3. Create the resource
4. View the Volume
5. View the PersistentVolumeClaim



Storage Classes

Introduction

A StorageClass is used by administrators to describe the "classes" of storage that they offer.



Kubernetes has no opinion about what classes represent.



StorageClass Resource

A StorageClass helps administrators describe various storage classes. Administrators can specify a default StorageClass for PVCs that are not bound to a particular class.

Demo

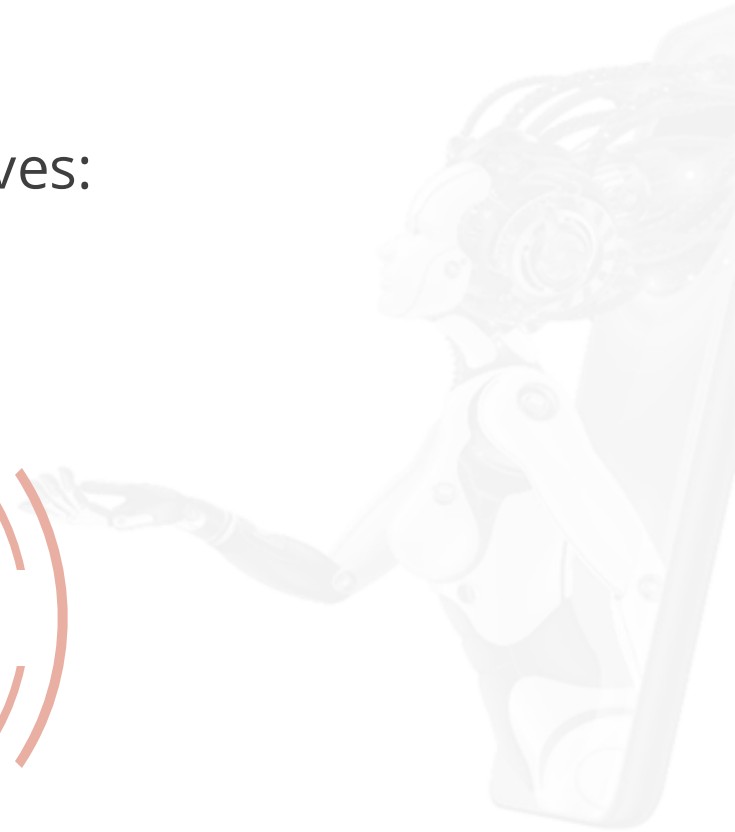
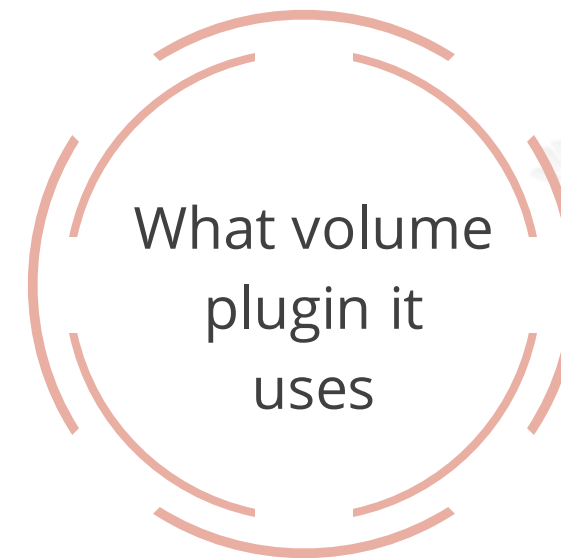
```
apiVersion: storage.k8s.io/v1
Kind: StorageClass
Metadata:
  name: standard
provisioner: kubernetes.io/aws-efs
parameters:
  type: gp2
reclaimPolicy: Retain
allowVolumeExpansion: true
mountOptions:
  - debug
volumeBindingMode: Immediate
```



Provisioner

Each StorageClass has a Provisioner that determines what Volume plugin is used for provisioning PVs.

Authors of external Provisioners have full discretion over how their code lives:



StorageClass Resource: Fields

Reclaim Policy

Persistent Volumes created dynamically by a StorageClass will have the Reclaim Policy specifics mentioned in the Reclaim Policy field of the class. It can be either Delete or Retain.

Volume Expansion

Configuration of Persistent Volumes can make them expandable.

Mount Options

Persistent Volumes that are dynamically created by a StorageClass will have the mount options specified in the mount Options field of the class.

Volume Binding Mode

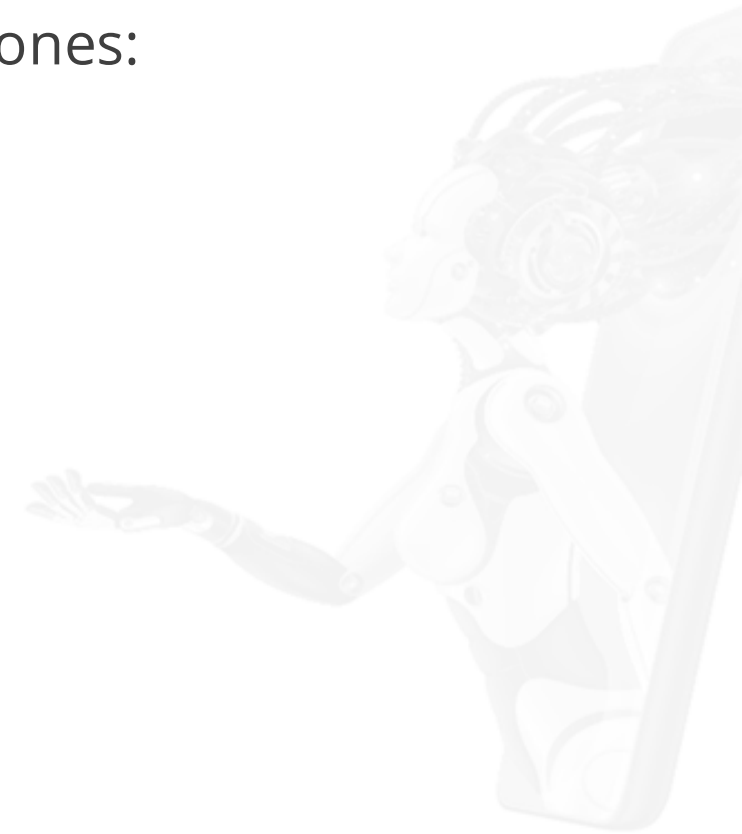
Volume Binding Mode controls when Volume Binding and Dynamic Provisioning should occur.

Allowed Topologies

When the WaitForFirstConsumer Volume Binding Mode is specified by a Cluster operator, provisioning to specific topologies need not be restricted. The code snippet below demonstrates the manner of restriction of topology of provisioned Volumes to specific zones:

Demo


```
apiVersion: storage.k8s.io/v1
Kind: StorageClass
Metadata:
  name: standard
provisioner: kubernetes.io/gce-pd
parameters:
  type: pd-standard
volumeBindingMode: WaitForFirstCustomer
allowedTopologies:
  - matchLabelExpressions:
    - key: failure-domain.beta.kubernetes.io/zone
      values:
      - vs-central-a
      - vs-central-b
```



Parameters

Storage classes have parameters that describe Volumes belonging to the StorageClass. A maximum of 512 parameters may be defined for a StorageClass.

The Provisioners include the following providers:

- 
- | | | | |
|---|------------------|---|-----------------|
| 1 | AWS EBS | 5 | vSphere |
| 2 | GCE PD | 6 | CSI Provisioner |
| 3 | Glusterfs | 7 | vCP Provisioner |
| 4 | OpenStack Cinder | 8 | Ceph RBD |

Parameters

9

Quo Byte

10

Azure Disk

11

Azure File

12

Portworx Volume

13

ScaleIO

14

StorageOS

15

Local

Understanding Storage Classes



Problem Statement:

Understand the working of Storage Classes in Kubernetes.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to demonstrate Storage Classes in Kubernetes:

1. Create a StorageClass
2. Create the resource
3. View the StorageClass



VolumeSnapshot Classes

Introduction

VolumeSnapshotClass can be used to describe the "classes" of storage when provisioning a VolumeSnapshot.



VolumeSnapshot Class Resource

Each VolumeSnapshotClass has fields for driver, Deletion Policy, and parameters.

Name and other parameters of a class can be finalized during the creation of VolumeSnapshotClass objects. Updating is not possible after creation.

Demo

```
apiVersion:  snapshot.storage.k8s.io/v1
Kind:       VolumeSnapshot
Metadata:
  name:      csi-hostpath-snapclass
  driver:    hostpath.csi.k8s.io
  deletionPolicy:  Delete
  parameters
```



VolumeSnapshot Class Resource

Default VolumeSnapshotClass can be specified for VolumeSnapshots that do not request any particular class to bind by adding the `snapshot.storage.kubernetes.io/is-default-class: "true"` annotation, as can be seen in the snippet below:

Demo

```
apiVersion: snapshot.storage.k8s.io/v1
Kind: VolumeSnapshot
Metadata:
  name: csi-hostpath-snapclass
  annotations:
    snapshot.storage.kubernetes.io/is-default-class: "true"
driver: hostpath.csi.k8s.io
deletionPolicy: Delete
parameters
```



VolumeSnapshot Class Resource

Driver

VolumeSnapshot classes have a driver to determine which CSI Volume plugin is used for provisioning VolumeSnapshots.

Deletion Policy

It enables configuration of a VolumeSnapshotContent when the VolumeSnapshot object it is bound to is to be deleted.

Parameters

VolumeSnapshot classes have parameters, which describe Volume Snapshots that belong to them.

Different parameters may be accepted, depending on the driver.



Understanding VolumeSnapshot Classes



Problem Statement:

Understand the working of Volume Snapshots classes in Kubernetes.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to demonstrate VolumeSnapshot Classes in Kubernetes:

1. Create a VolumeSnapshot class
2. Create the configuration
3. Create the resource
4. View the VolumeSnapshot classes



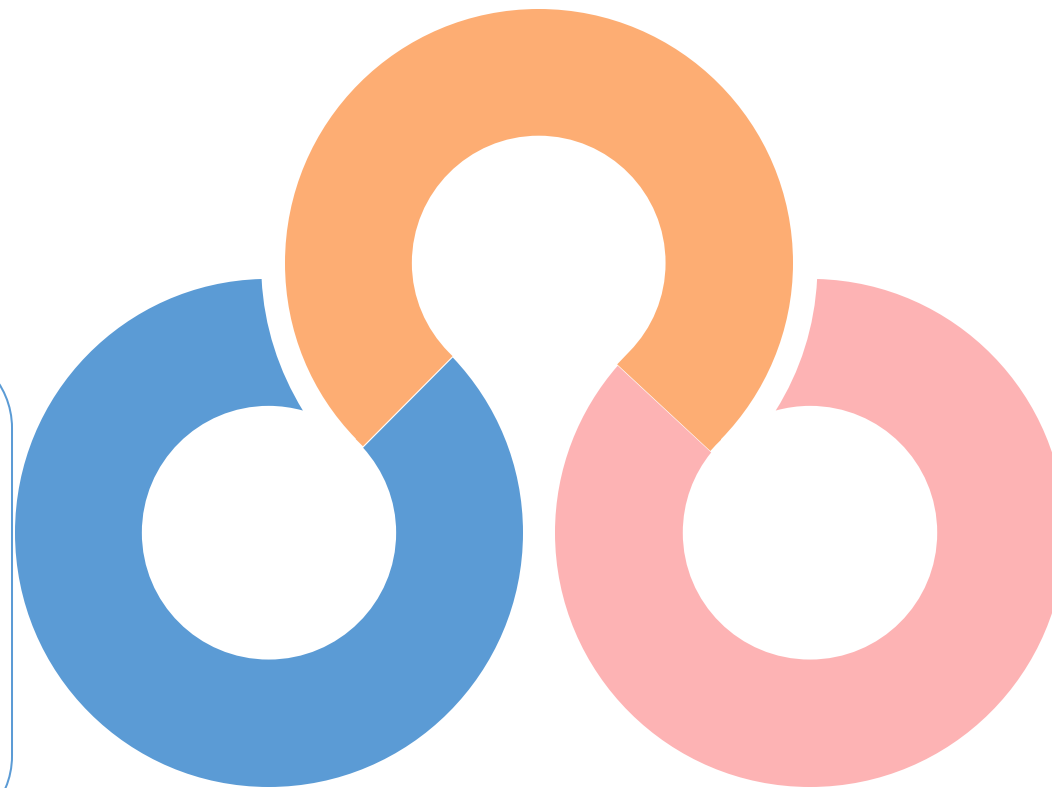
Dynamic Volume Provisioning

Overview

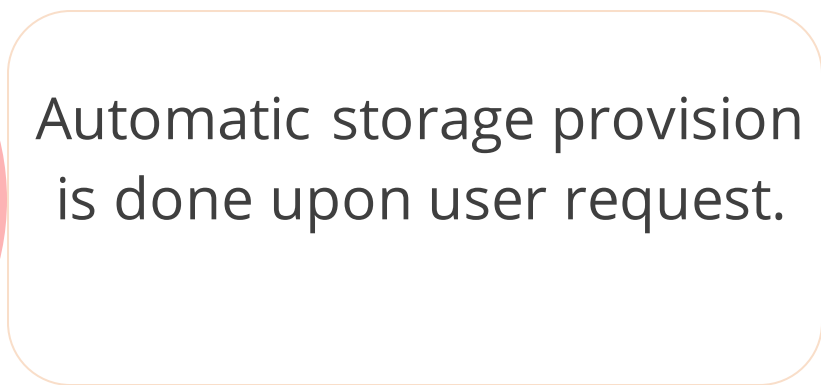
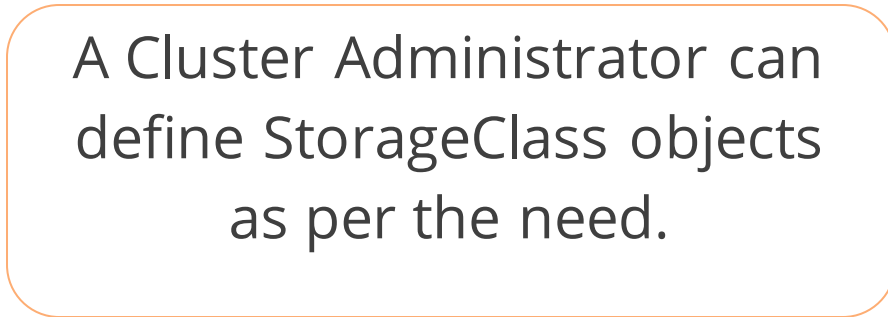
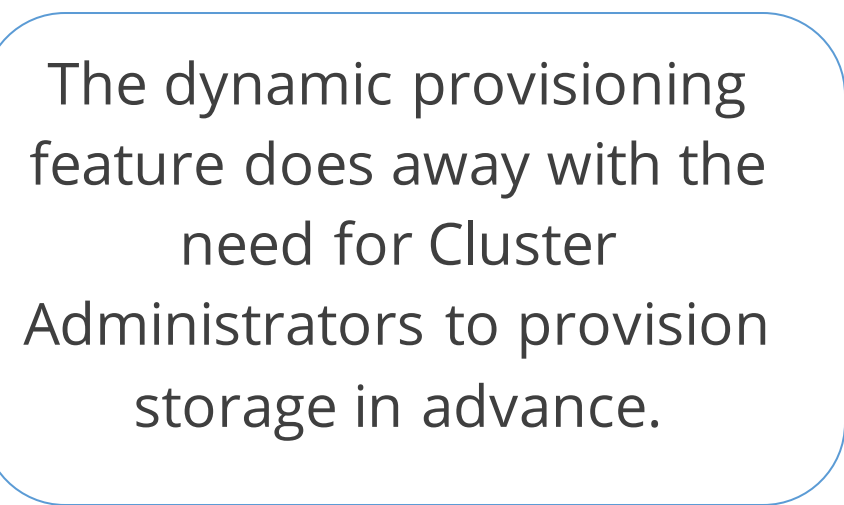
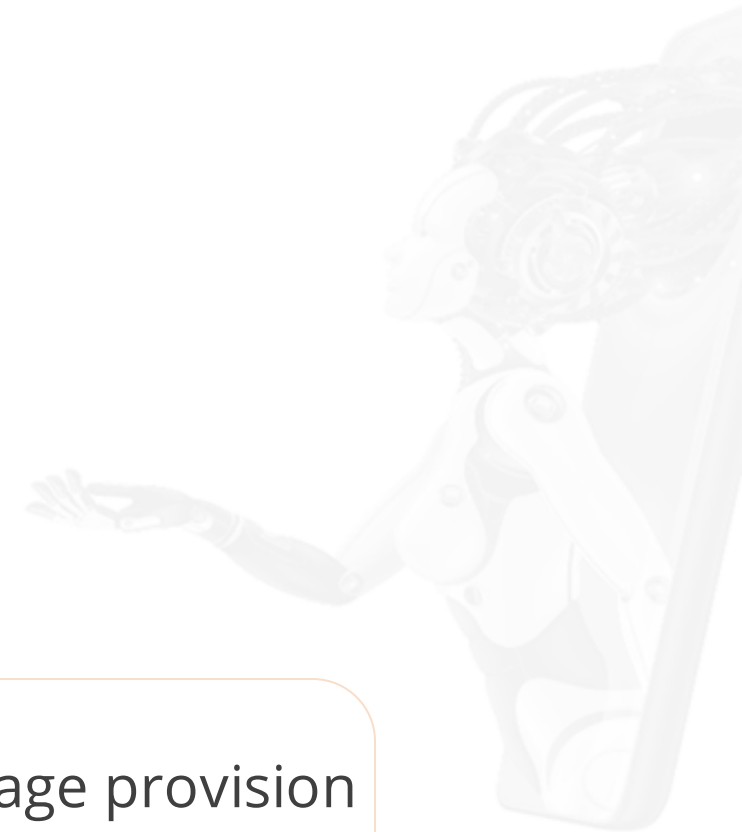
Dynamic Volume Provisioning allows Storage Volumes to be created on-demand.

A Cluster Administrator can define StorageClass objects as per the need.

The dynamic provisioning feature does away with the need for Cluster Administrators to provision storage in advance.



Automatic storage provision is done upon user request.



Enabling Dynamic Provisioning

To enable Dynamic Provisioning, a Cluster Administrator needs to create, in advance, one or more StorageClass objects.

Here are two storage classes being created: provisions standard disk-like persistent disks and SSD-like persistent disks.

Demo

```
apiVersion: storage.k8s.io/v1
Kind: StorageClass
Metadata"
  name: slow
provisioner: kubernetes.io/gce-pd
parameters:
  type: pd-standard
```

Demo

```
apiVersion: storage.k8s.io/v1
Kind: StorageClass
Metadata"
  name: fast
provisioner: kubernetes.io/gce-pd
parameters:
  type: pd-ssd
```

Usage of Dynamic Provisioning

Before Kubernetes v1.6, `volume.beta.kubernetes.io/storage-class` annotation was used. For "fast" StorageClass, a user would create the PersistentVolumeClaim as shown here:

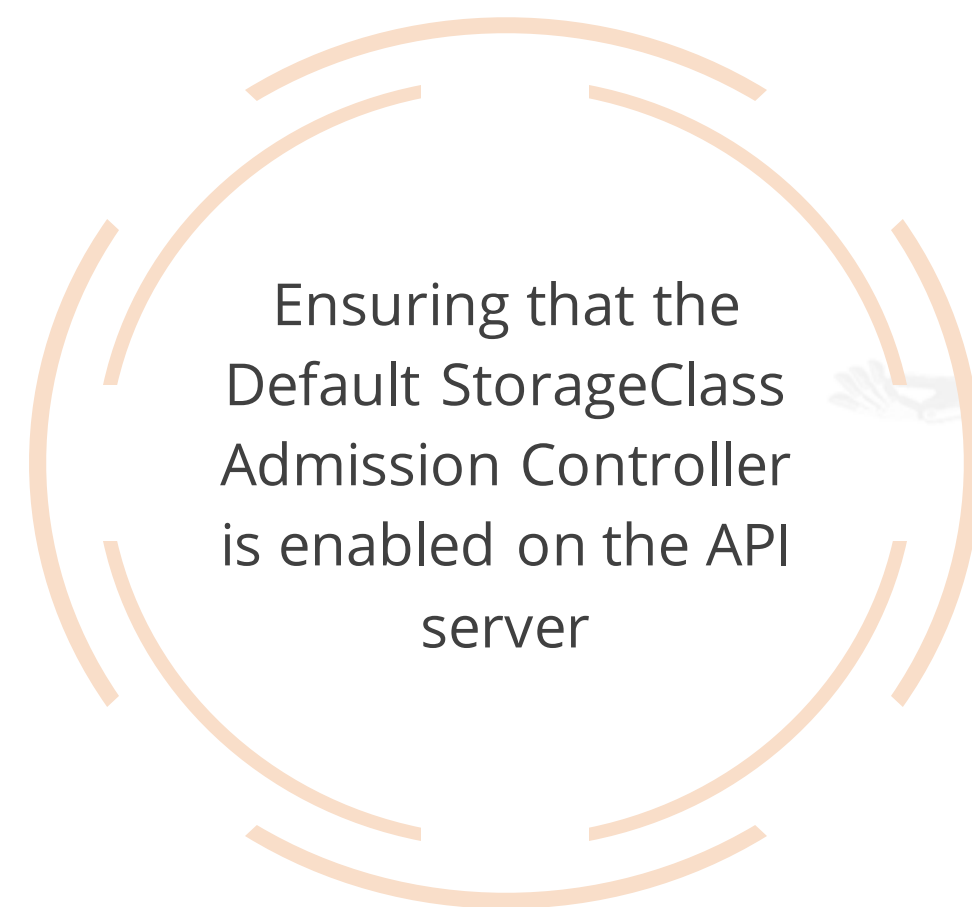
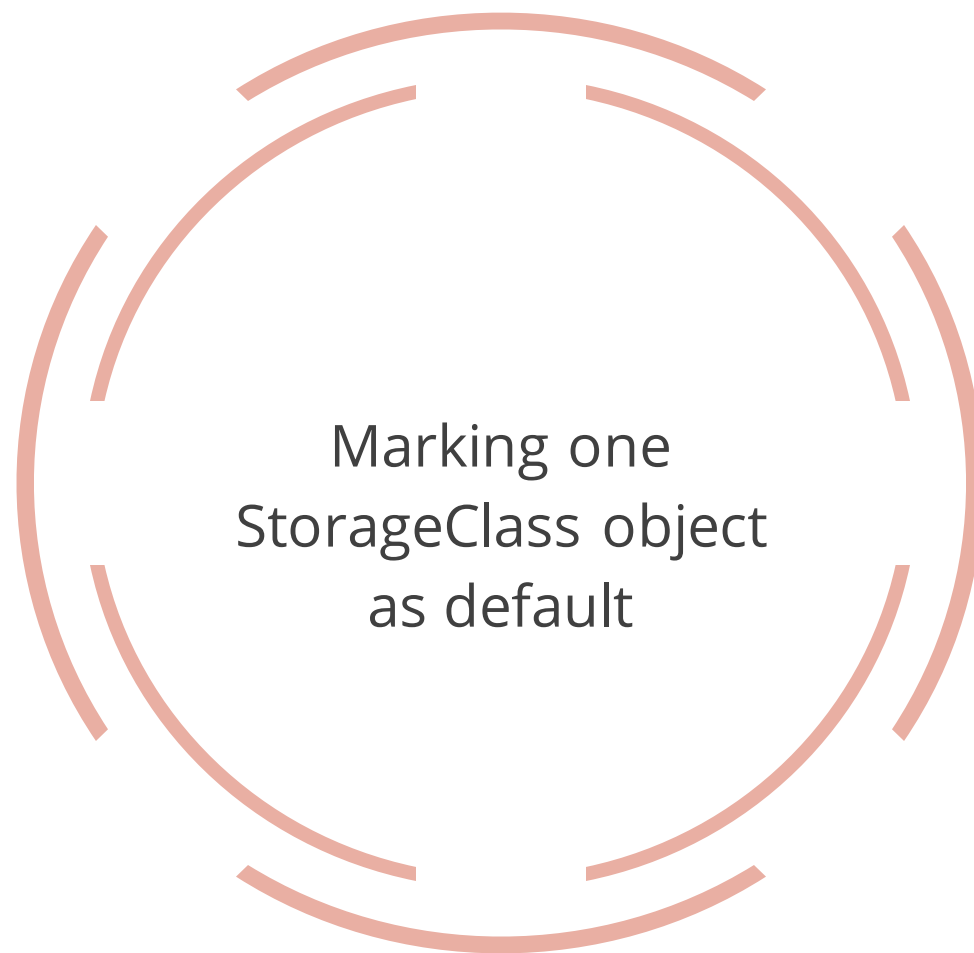
Demo

```
apiVersion: v1
Kind: PersistentVolumeClaim
Metadata:
  name: claim1
Spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: fast
  resources:
    requests:
      storage: 30Gi
```



Defaulting Behavior

This is how a Cluster administrator enables Defaulting Behavior.



Topology Awareness

In Multi-zone Clusters, Pods can be spread across Zones in a Region.



Single-zone storage backends should be provisioned in the Zones where Pods are scheduled.

Understanding the Working of Dynamic Volume Provisioning



Problem Statement:

Understanding the working of dynamic volume provisioning in Kubernetes.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to demonstrate Dynamic Volume Provisioning in Kubernetes:

1. Enable Dynamic Provisioning in StorageClass
2. Use Dynamic Provisioning in PersistentVolumeClaim



Storage Capacity

Storage Capacity

Storage Capacity is limited and may vary, depending on the Node on which a Pod runs.

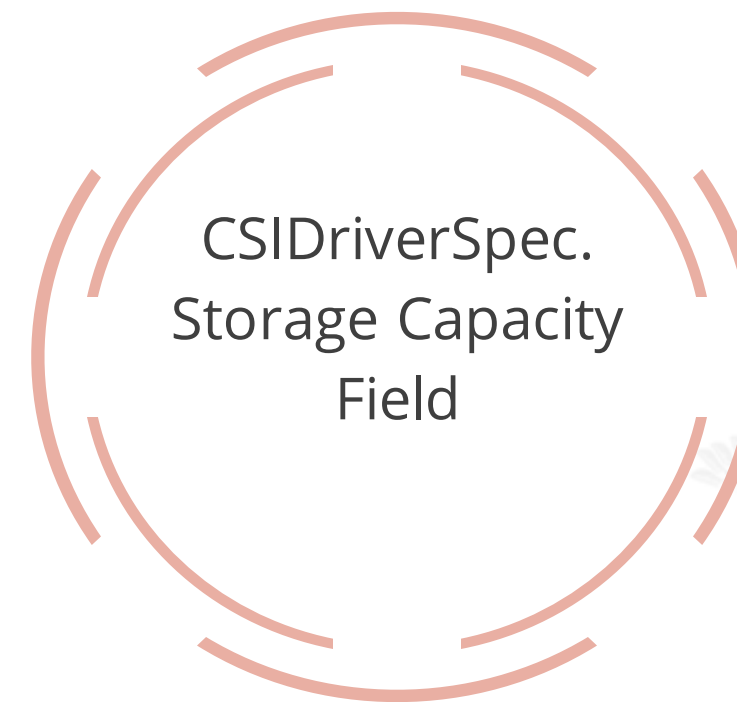
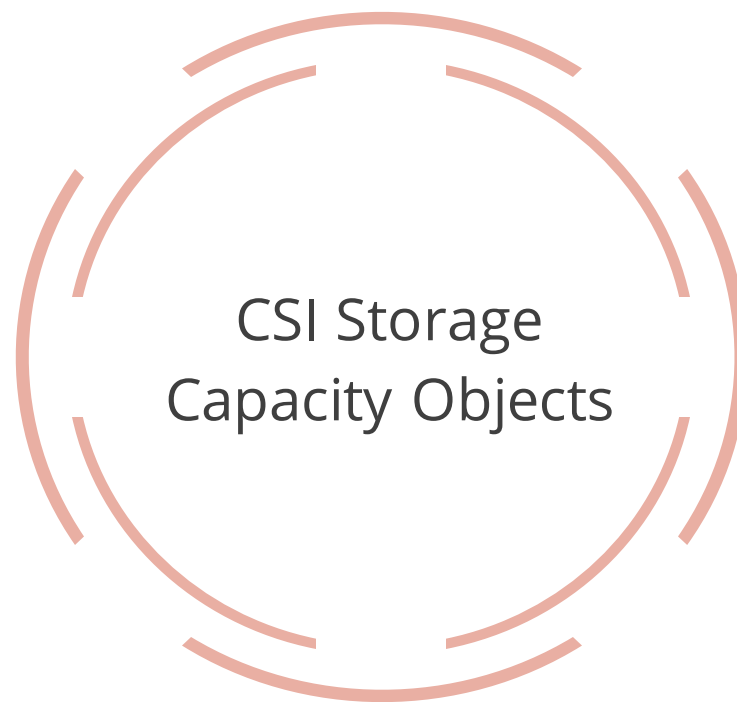


Tracking Storage Capacity is supported for Container Storage Interface (CSI) drivers and should to be enabled when a CSI driver is installed.



API

There are two API extensions for this feature:



Criteria Scheduling

Scenarios when Storage Capacity information is used by the Kubernetes Scheduler:



The CSIStorageCapacity feature gate is true



A Pod uses a Volume that has not been created



Volume uses a StorageClass which references a CSI driver and uses WaitForFirstConsumer Volume Binding Mode



The CSIDriver object for the driver has StorageCapacity is set to true



Scheduling

It compares the size of the volume against the capacity listed in CSI Storage Capacity objects with a topology that includes the Node.

For Volumes with Immediate Volume Binding Mode, the Volume creation location is decided by the Storage Driver.

For CSI Ephemeral Volumes, scheduling does not take Storage Capacity into consideration.



Rescheduling

After a Node is selected for a Pod with WaitForFirstConsumer Volumes, the CSI Storage Driver is asked to create the Volume with a specification that the Volume should be available on the selected Node.

Storage Capacity tracking increases the chance of scheduling working on the first try.

When a Pod uses multiple volumes, scheduling can fail permanently.

Enable Storage Capacity Tracking

Storage Capacity tracking is an alpha feature. It functions only when the CSIStorageCapacity feature gate and the storage.k8s.io/v1alpha1 API groups are enabled. The code snippet given here may be used to list CSIStorageCapacity objects.

Demo

```
#A quick check whether a Kubernetes cluster supports the feature is to list  
CSIStorageCapacity objects with:
```

```
kubectl get csistoragecapacities --all-namespaces
```

```
#If the cluster supports CSIStorageCapacity, the response is either a list of  
CSIStorageCapacity objects or:
```

```
No resources found
```

```
#If not supported, this error is printed instead:
```

```
error: the server doesn't have a resource type "csistoragecapacities"
```

Tracking Storage Capacity



Problem Statement:

Learn how to track the storage capacity in Kubernetes.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to demonstrate Storage Capacity in Kubernetes:

1. Track storage capacity
2. Type ***kubectrl get csistoragecapacities -A***



Ephemeral Volumes

Overview

Caching services often have limited memory size. They can shift infrequently-used data into Storage.

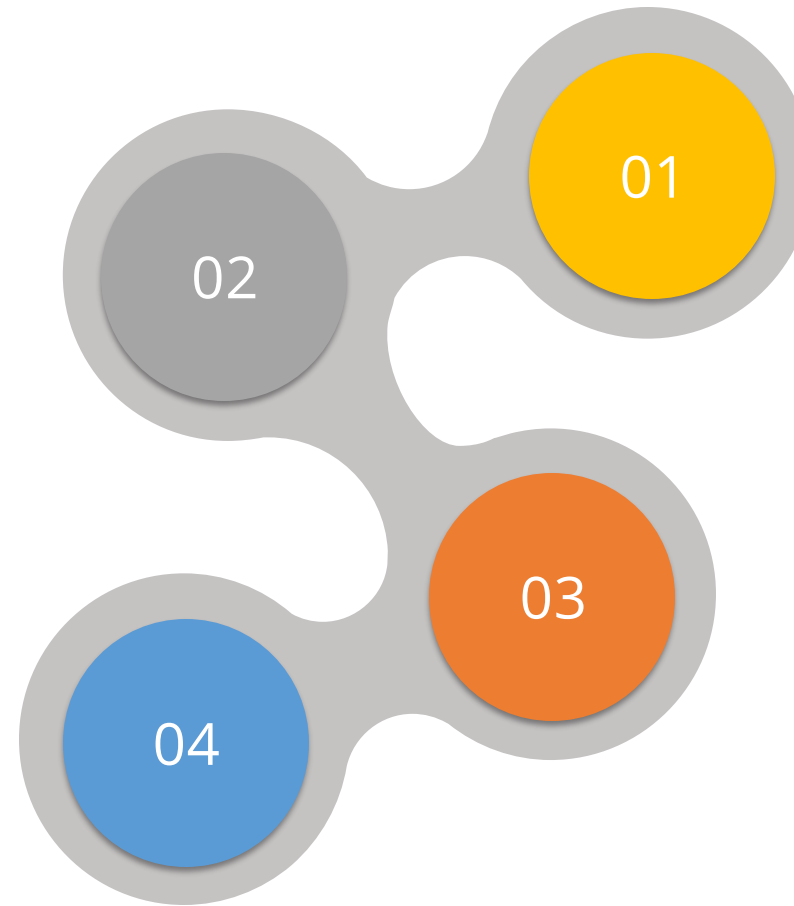
Ephemeral Volumes are designed for specific use cases.

They are specified inline in the Pod spec. This makes application deployment and management simple.



Types of Ephemeral Volumes

Generic Ephemeral Volumes



emptyDir

CSI Ephemeral Volumes

configMap, downwardAPI, secret

Third-party CSI Storage Drivers must provide Ephemeral Volumes.

Ephemeral Volumes

Third-party CSI Storage Drivers and other Storage Drivers that support Dynamic Provisioning may provide Generic Ephemeral Volumes.

Third-party drivers offer some functionality that is not supported by Kubernetes.

Some CSI drivers are written specifically for CSI Ephemeral Volumes and do not support Dynamic Provisioning.



They cannot be used for Generic Ephemeral Volumes.

CSI Ephemeral Volumes

CSI Ephemeral Volumes are managed locally on each Node. They are created along with other local resources after a Pod is scheduled on a Node. Here is an example manifest for a Pod that uses CSI Ephemeral Storage:

Demo

```
Kind: Pod
apiVersion: v1
Metadata:
  name: my-csi-app
Spec:
  Containers:
    - name: my-frontend
      image: busybox
      volumeMounts:
        - mountPath: "/data"
          name: my-csi-inline-vol
      command: [ "sleep", "1000000" ]
  volumes:
    - name: my-csi-inline-vol
      csi:
        driver: inline.storage.kubernetes.io
        volumeAttributes:
          foo: bar
```



Generic Ephemeral Volumes

Generic Ephemeral Volumes are just like emptyDir volumes. Here is a manifest of a Generic Ephemeral Volume:

Demo

```
Kind: Pod

  apiVersion: v1
  Metadata:
    name: my-app
  Spec:
    Containers:
      - name: my-frontend
        image: busybox
        volumeMounts:
          - mountPath: "/scratch"
            name: scratch-volume
        command: [ "sleep","1000000" ]
    volumes:
      - name: scratch-volume
        ephemeral:
```

Demo

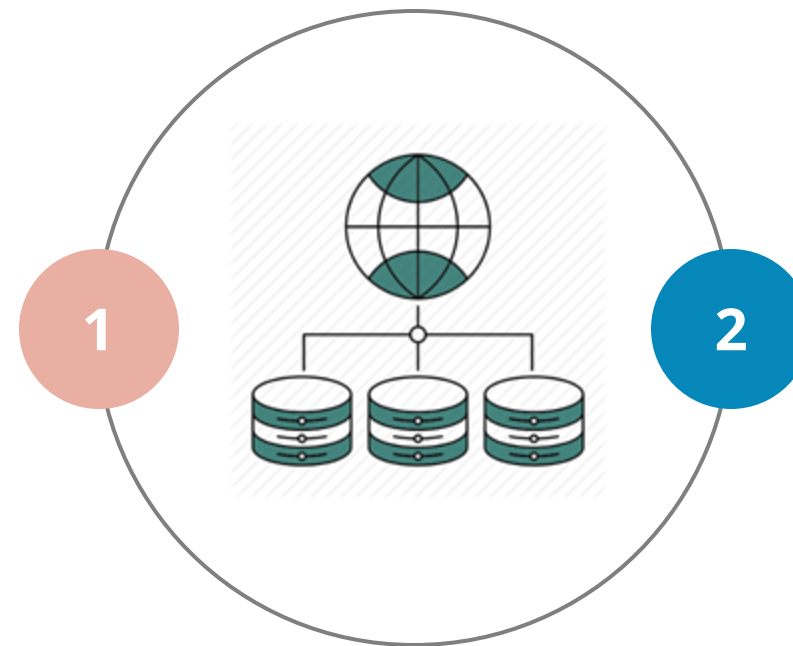
```
  volumeClaimTemplate:
    Metadata:
      Labels:
        type: my-frontend-volume
    spec: inline.storage.kubernetes.io
      accessModes: [ "ReadWriteOnce" ]
      storageClassName: "scratch-storage-
class"

    Resources:
      requests:
        Storage: 1Gib
```

Life cycle and PersistentVolumeClaim

The key design idea is that the parameters for a Volume Claim are allowed inside a Volume source of the Pod.

In terms of resource ownership, a Pod that has Generic Ephemeral Storage is the owner of the PersistentVolumeClaim(s) that provide(s) the Ephemeral Storage.



The Kubernetes Garbage Collector deletes the PVC when the Pod is deleted. This action triggers deletion of the Volume. The default Reclaim Policy of Storage Classes is to delete Volumes.

PersistentVolumeClaim Naming

Names of the automatically created PVCs are a combination of Pod name and Volume name, joined with a hyphen (-).

1

PVC is used for an Ephemeral Volume only if it was created for the Pod.

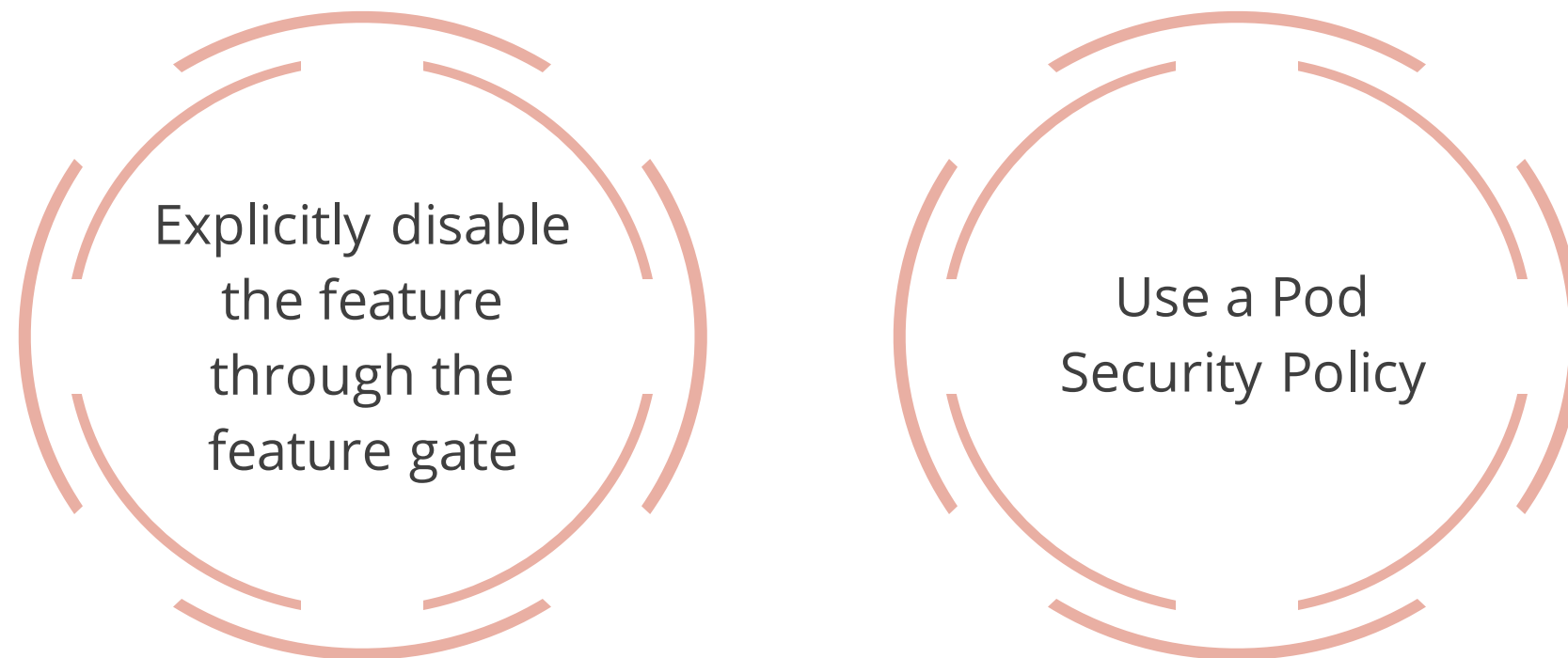
2

The deterministic naming could result in a potential conflict between Pods, and between Pods and manually created PVCs.

Security

Enabling the GenericEphemeralVolume feature allows users to create PVCs indirectly if they can create Pods, even if they do not have permission to create PVCs directly.

Two choices, if this does not fit the security model, are:



Understanding Ephemeral Volumes



Problem Statement:

Understand the working of Ephemeral Volumes in Kubernetes.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to demonstrate Ephemeral Volumes in Kubernetes:

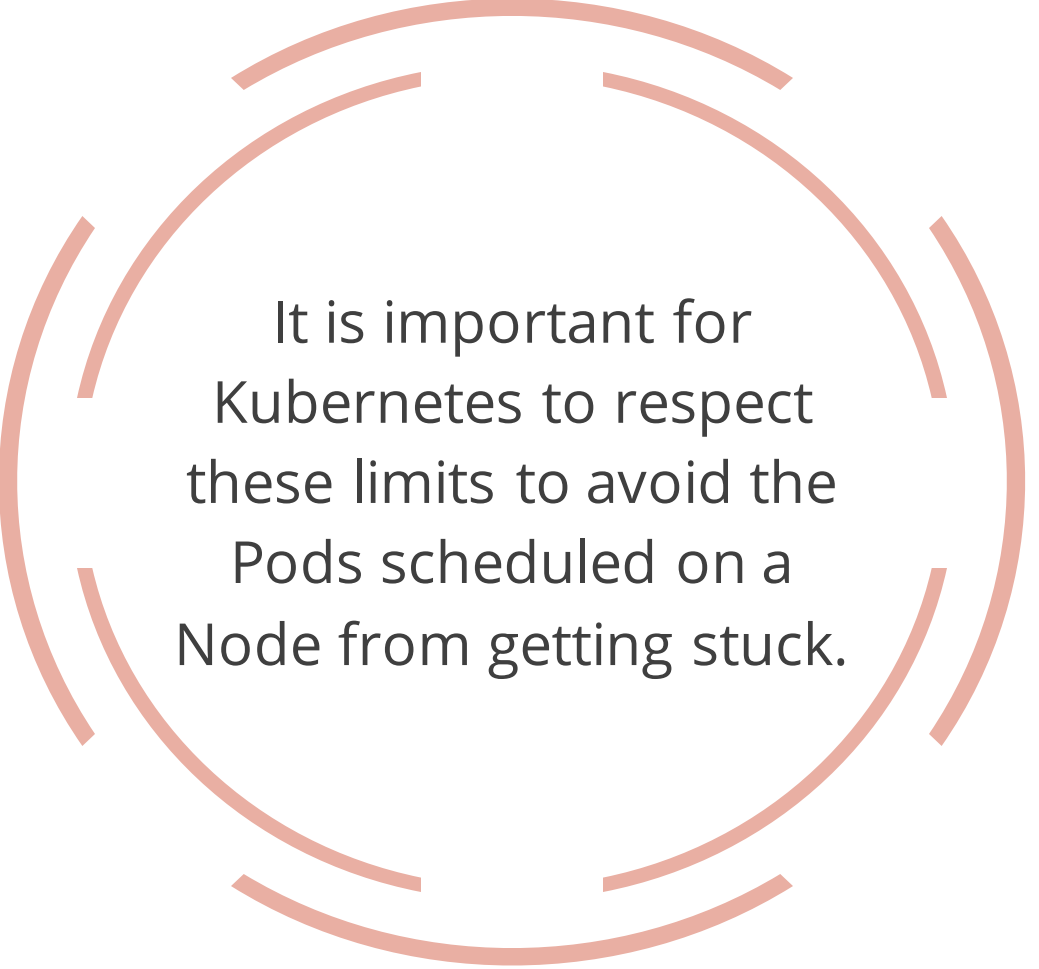
1. Create a Pod for CSI Ephemeral Volumes
2. Create the configuration
3. Create the resource
4. View the VolumeSnapshot classes



Node-Specific Volume Limits

Introduction

Cloud providers like Google, Amazon, and Microsoft typically have a limit on how many Volumes can be attached to a Node.



It is important for
Kubernetes to respect
these limits to avoid the
Pods scheduled on a
Node from getting stuck.



Kubernetes Default Limits

The Kubernetes Scheduler has default limits on the number of Volumes that can be attached to a Node.

Cloud service	Maximum Volumes per Node
Amazon Elastic Block Store (EBS)	39
Google Persistent Disk	16
Microsoft Azure Disk Storage	16

Custom Limits

Limits can be changed by setting the value of the KUBE_MAX_PD_VOLS environment variable, and then starting the Scheduler.

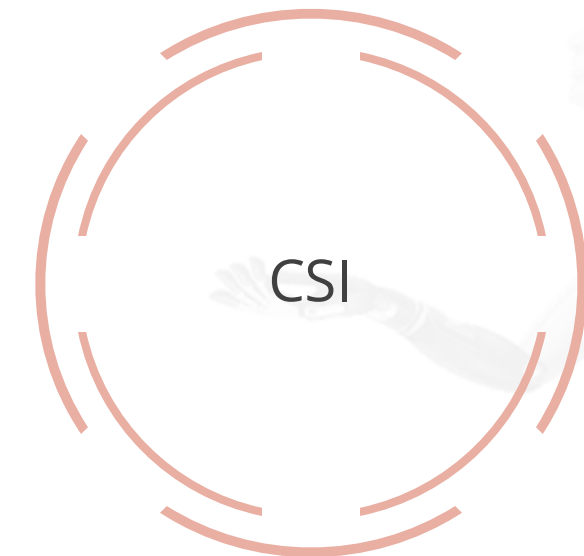
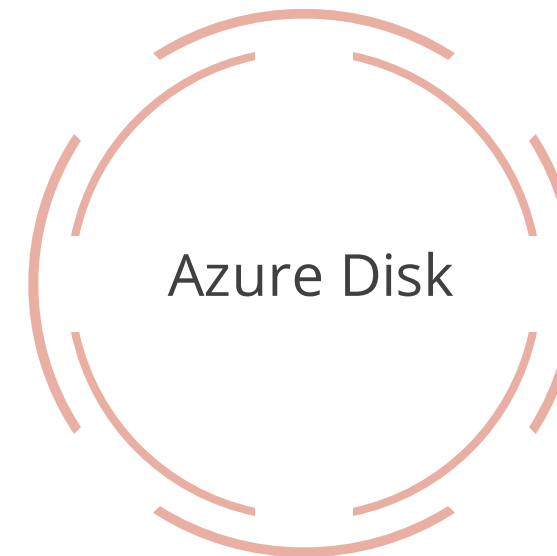
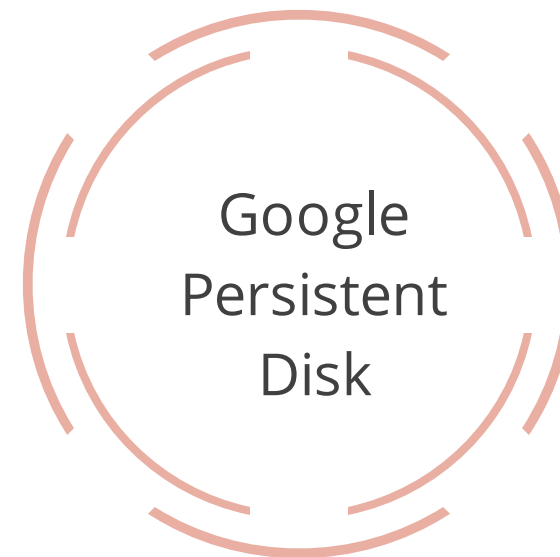


Use caution if you set a limit that is higher than the default limit.



Dynamic Volume Limits

Dynamic Volume Limits are supported for the following:



Key Takeaways

- A Pod can use any number of Volumes and Volume types simultaneously while working with Kubernetes.
- A StorageClass provides a way for administrators to describe the "classes" of storage that they offer.
- CSI Volume Cloning feature adds support for specifying existing PVCs in the dataSource field to indicate that a user would like to clone a Volume.
- Tracking Storage Capacity is supported for Container Storage Interface (CSI) drivers and needs to be enabled while installing a CSI driver.



Configuring Cluster Storage

Problem Statement:

You are a DevOps professional with expertise in Kubernetes. Your manager has asked you to create a new microservice and allot dedicated storage for it. The volume must have the provision of adding storage, if necessary, without involving the cluster administrator.

Steps to Perform:

1. Creating a Persistent Volume
2. Provisioning Volume dynamically

