

TECHNOLOGY



Container Orchestration using Kubernetes

TECHNOLOGY

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



A Day in the Life of a DevOps Engineer

You are working as a DevOps Engineer in an organization, and you've been assigned a task to deploy WordPress and MySQL using persistent volume and persistent volume claim. Your organization needs WordPress and MySQL application.

WordPress must be deployed using host path and MySQL must be deployed using NFS. The goal is to configure dynamic volume provisioning, secrets in volumes, CSI volume cloning, and tracking storage capacity. You are also required to limit the volumes to the specific nodes.

To achieve all the above, along with some additional features, we would be learning a few concepts in this lesson that will help you find a solution for the above scenario.



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. This can also be done for data for various applications.

Administrators can assign “classes” of various storage-to-map service quality levels by using **StorageClass**.

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



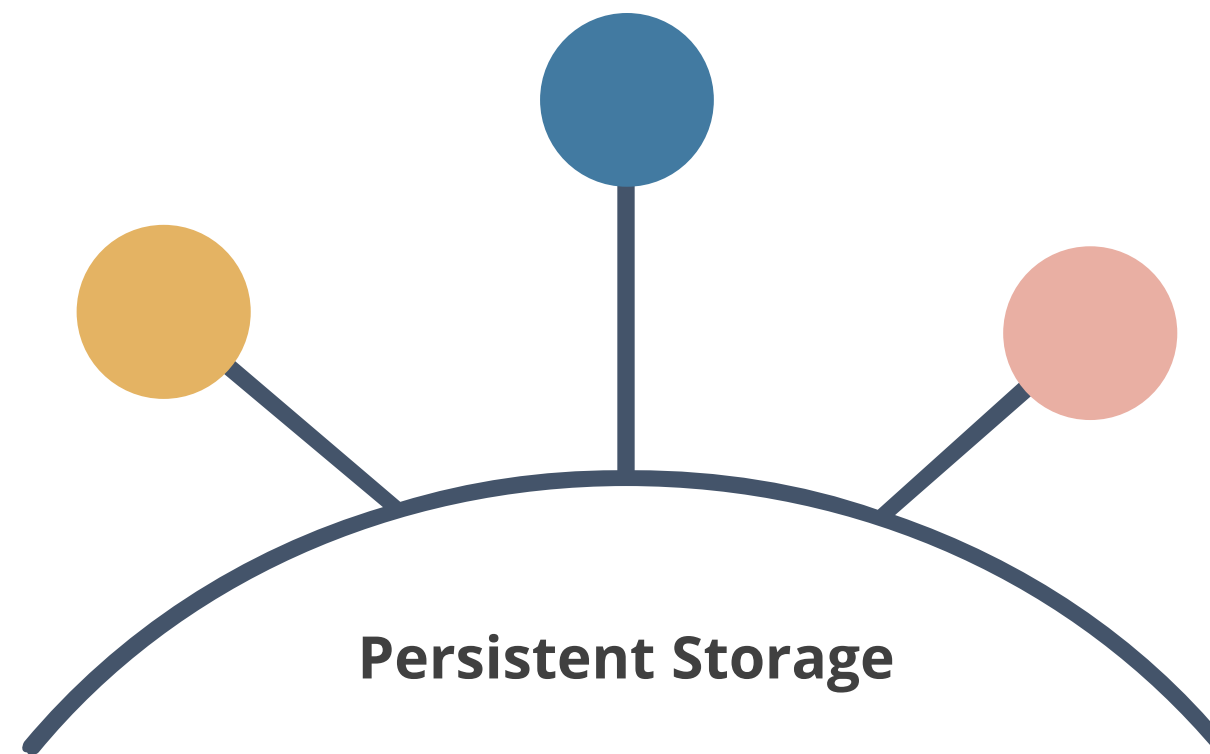
Kubernetes and Persistent Storage

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 a deletion of localized data.

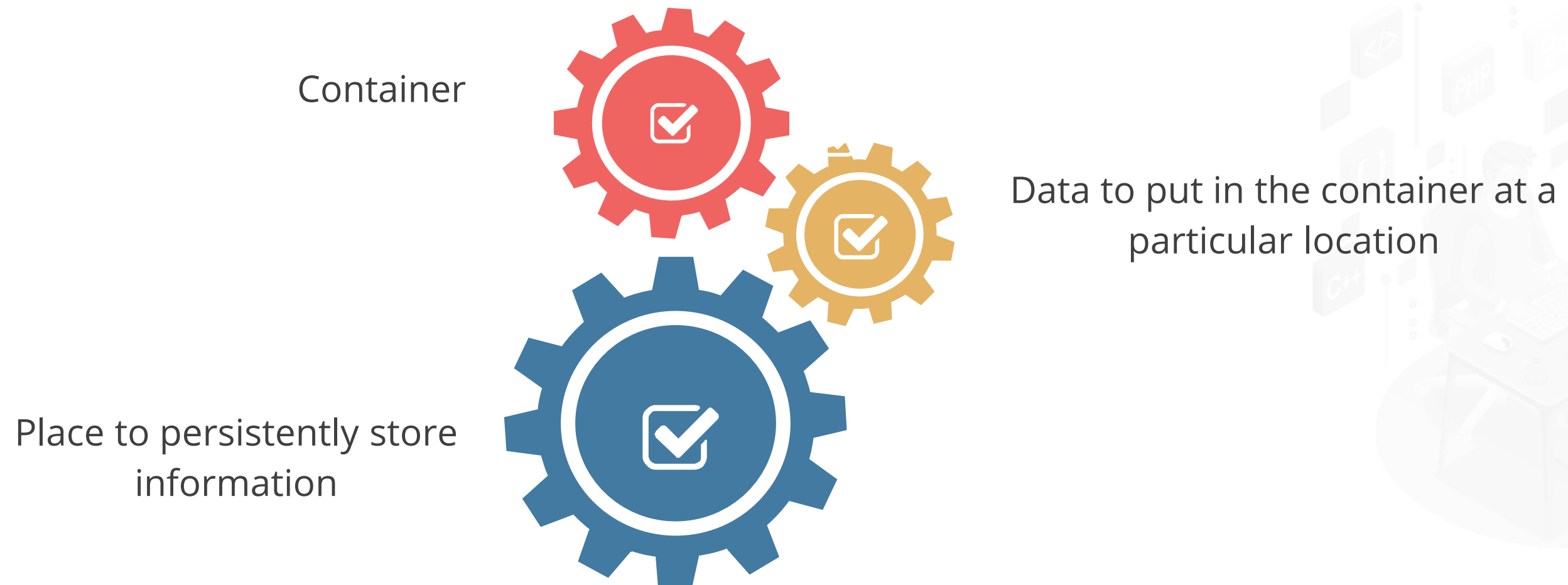
Applications need to remain functional and be in their unaltered state.

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



Requirements for Persistent Storage

These are the 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.

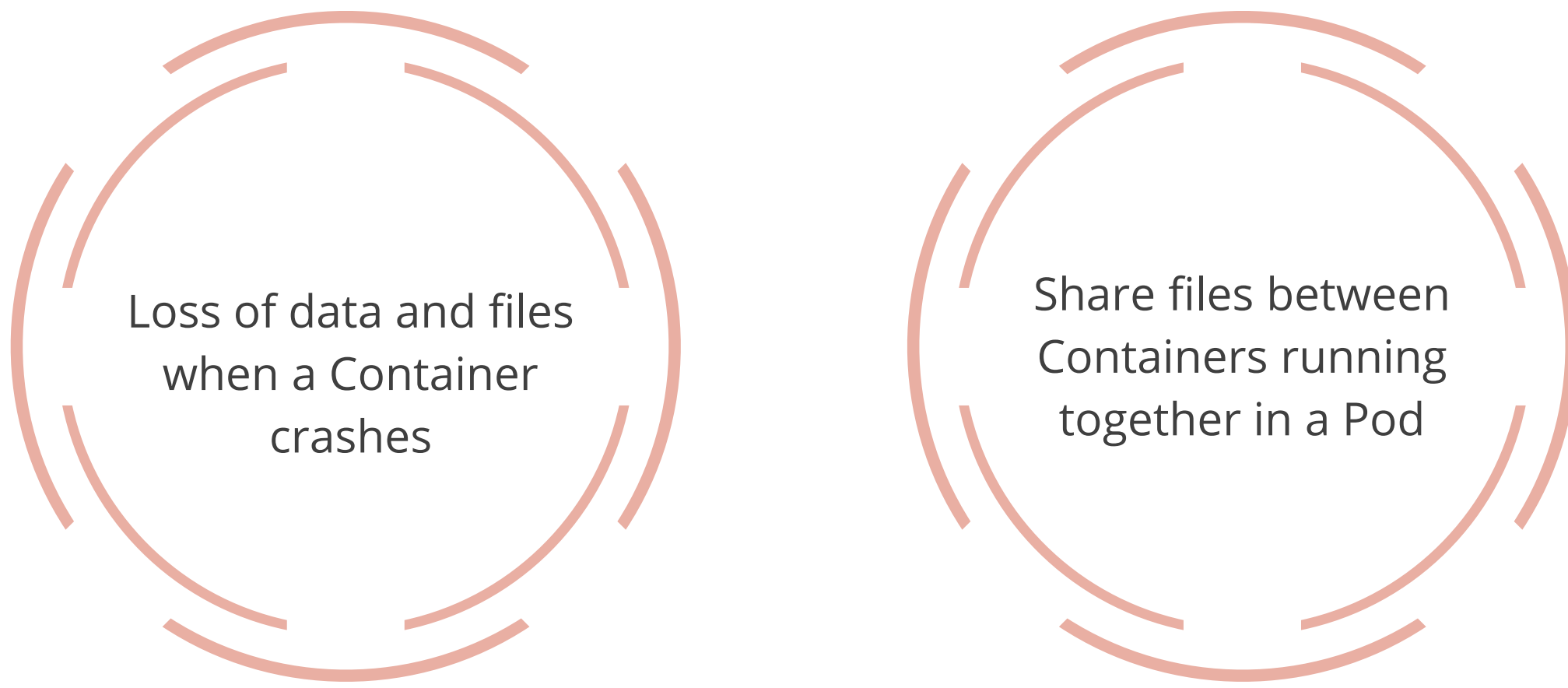


TECHNOLOGY

Volumes

Overview

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



Loss of data and files
when a Container
crashes

Share 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:

awsElasticBlockStore

azureDisk

azureFile

cephfs

cinder

configMap

downwardAPI

emptyDir

fibre channel

gcePersistentDisk

glusterfs

hostPath

Types of Volumes

Kubernetes supports many 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.

EBS Volume is not mounted, and its contents are persistent.



EBS Volume can be populated in advance with data, and data sharing between pods is possible.

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 the user's Pod.



cephfs volume is unmounted and its contents are preserved.



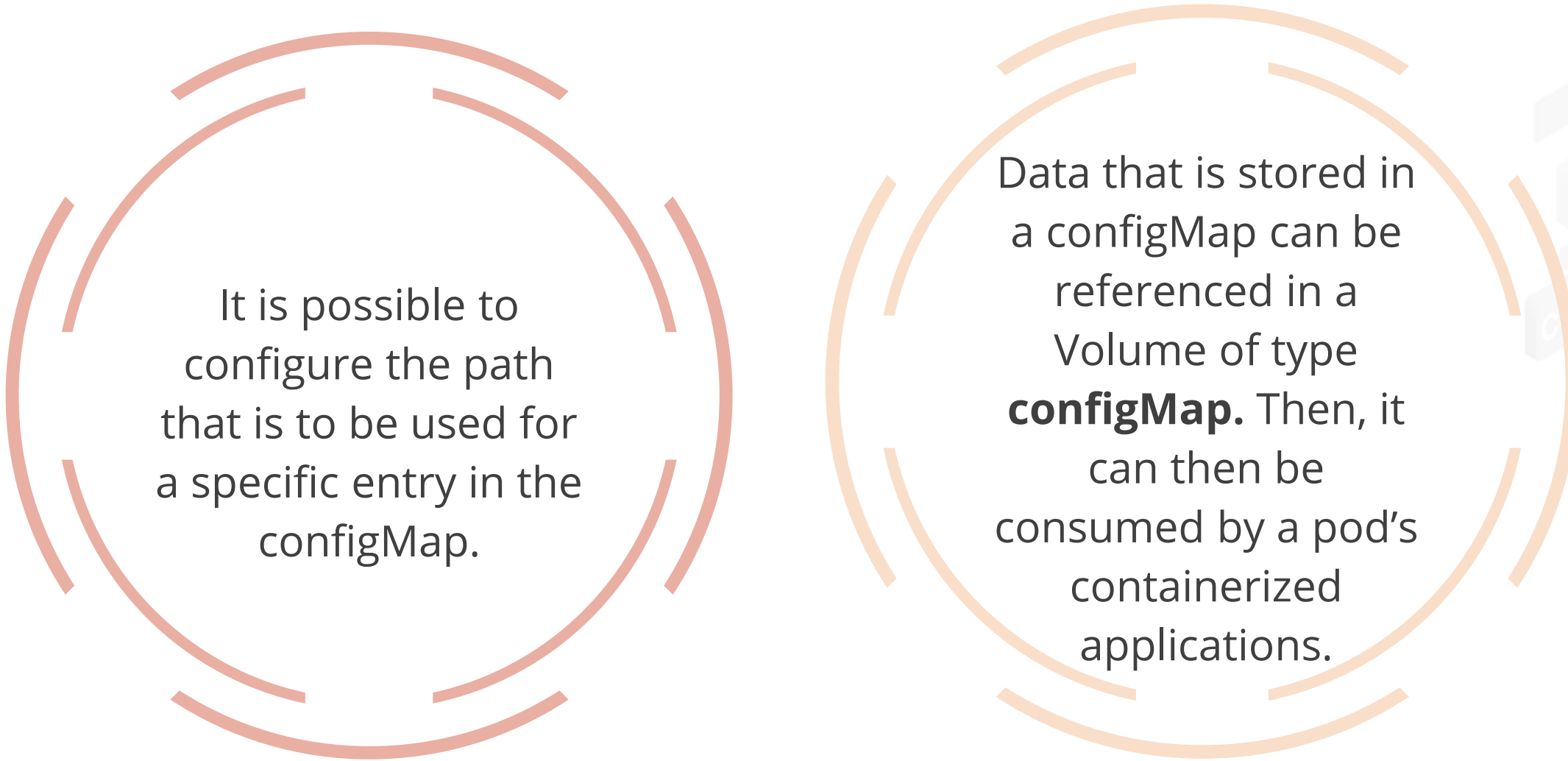
cephfs volume can be prepopulated with data.



Multiple writers can simultaneously mount onto a cephfs volume.

ConfigMap

ConfigMaps 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**. Then, 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:

- ✓ It provides scratch space, such as a disk-based merge sort
- ✓ Checkpoints are a long computation for recovery from crashes
- ✓ It holds files of content-manager Container at the time of the output from web server Container

gcePersistentDisk and iscsi

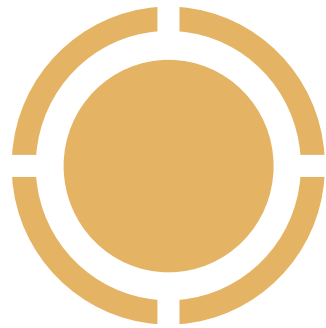
A **gcePersistentDisk** volume mounts a Google Compute Engine (GCE) Persistent Disk (PD) onto the Pod.

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

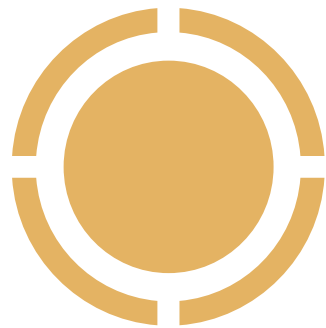
An **iscsi** volume allows an existing iSCSI (SCSI over IP) volume to be mounted onto the Pod.

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

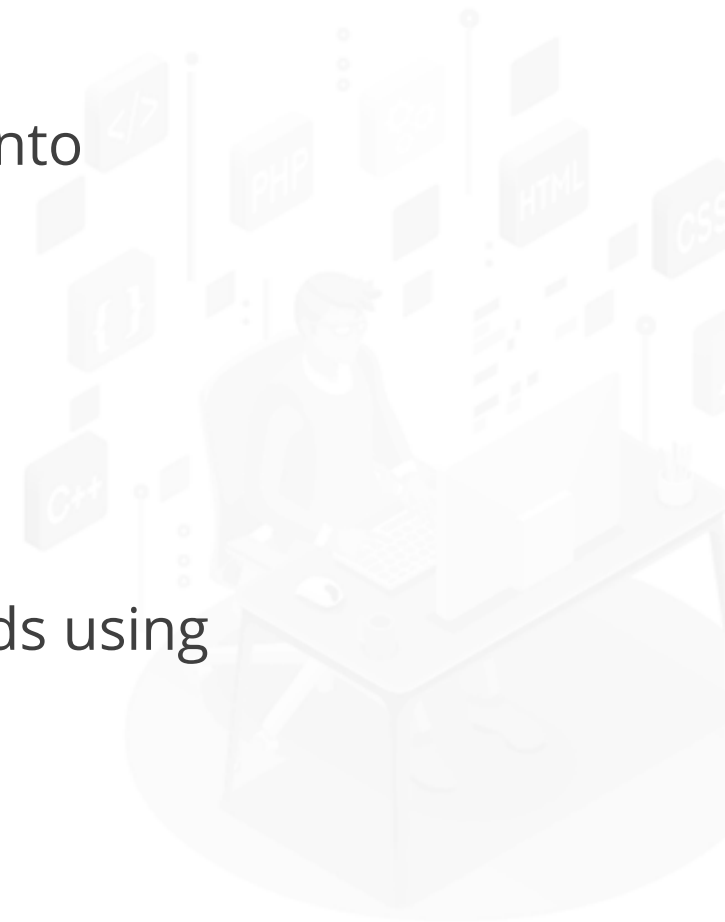
NFS and Secret



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



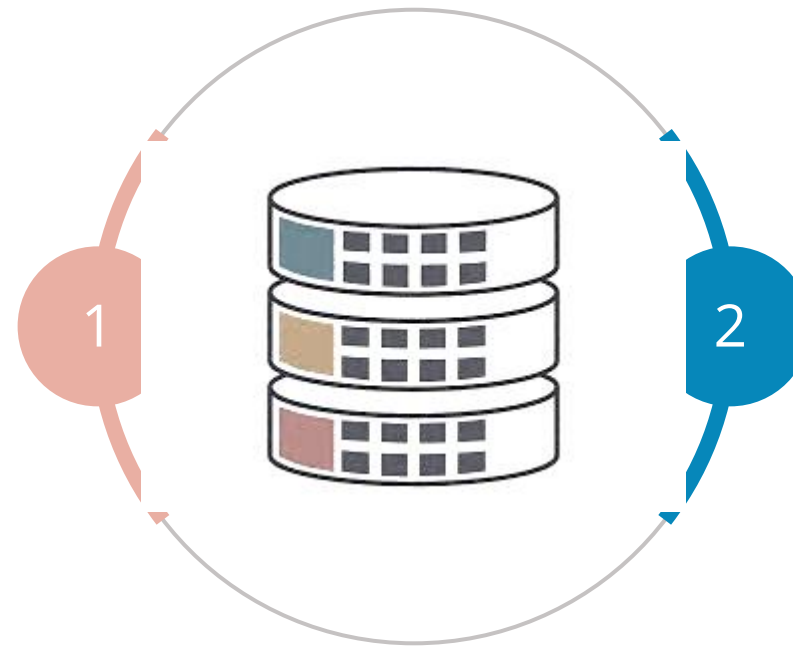
Sensitive information, such as passwords, can be passed to Pods using a secret volume.



StorageOS

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

StorageOS replicates data and create protection against node failure.



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

Using a hostPath



Duration: 15 mins

Problem Statement:

You've been asked to create a hostPath volume to mount files from a Pod to the file system of the host node.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed:

1. Creating a Pod using hostPath
2. Creating files within the Pod
3. Accessing the files created on other nodes

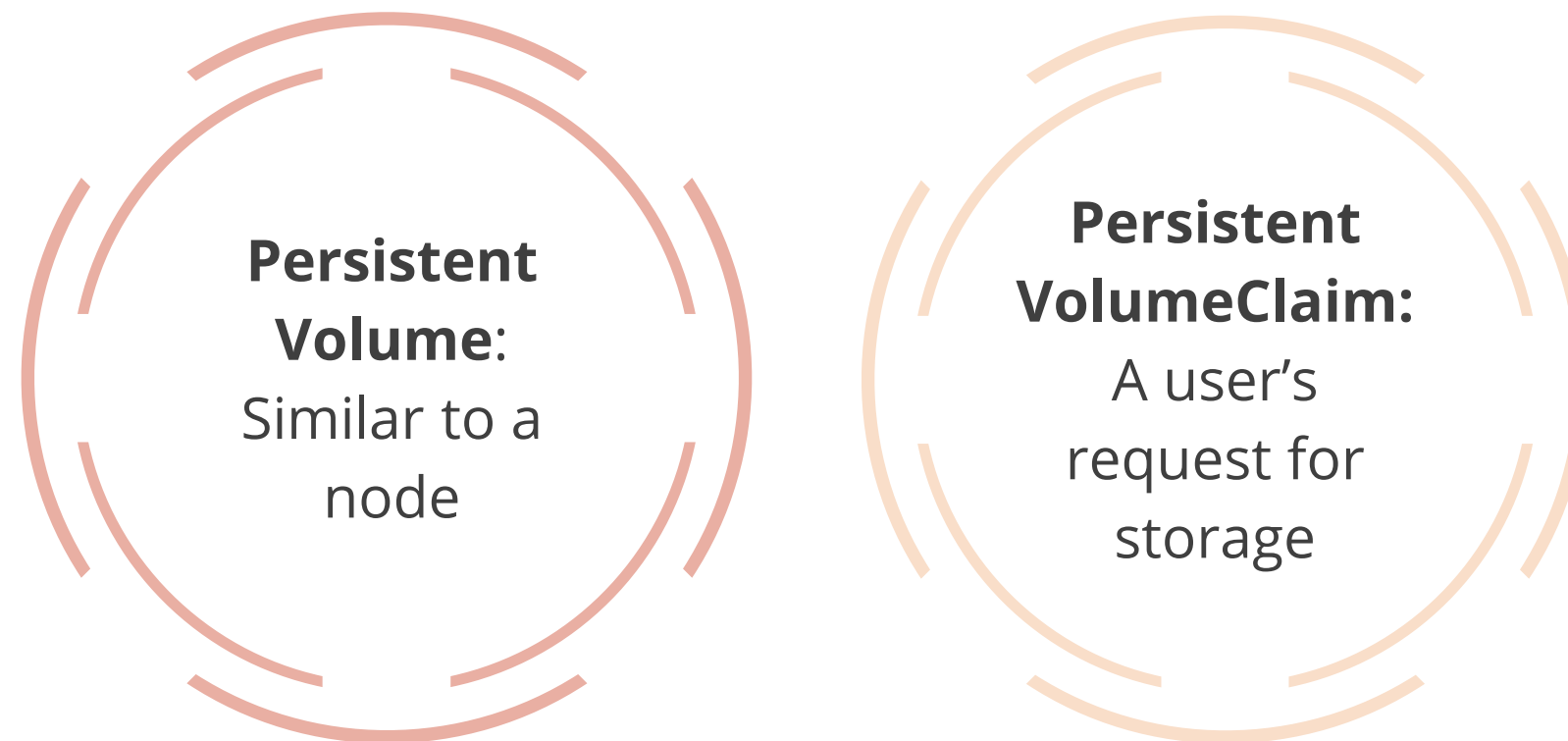


Persistent Volumes

Persistent Volumes

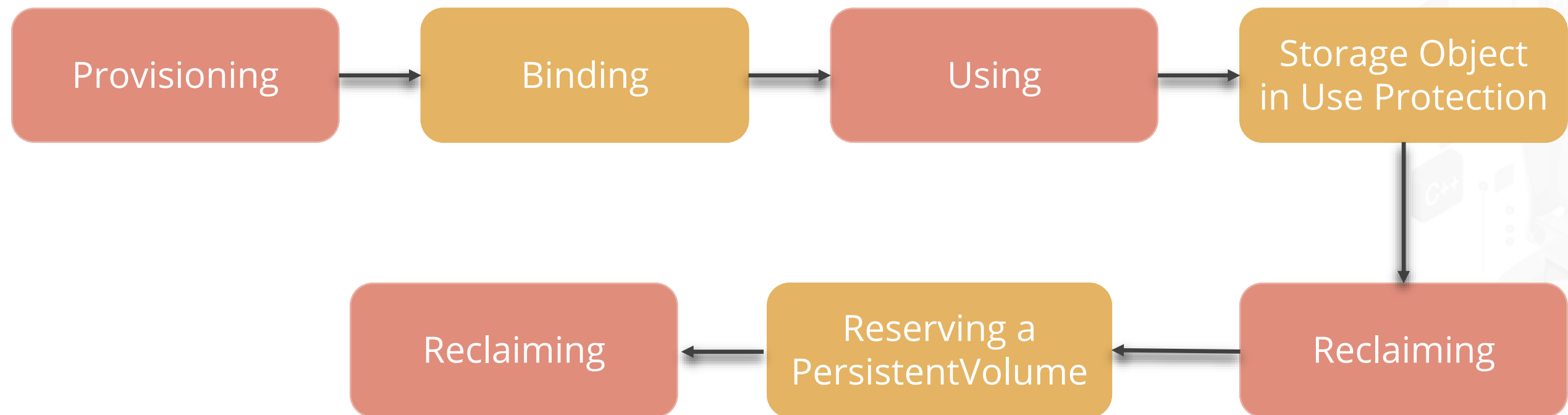
Users and administrators get an API from the **PersistentVolume** subsystem in the Kubernetes orchestration system. This API abstracts storage-related details.

Kubernetes technology introduces two new API resources:



Lifecycle of Volume and Claim

The lifecycle 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

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

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

These are the specifications used 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.

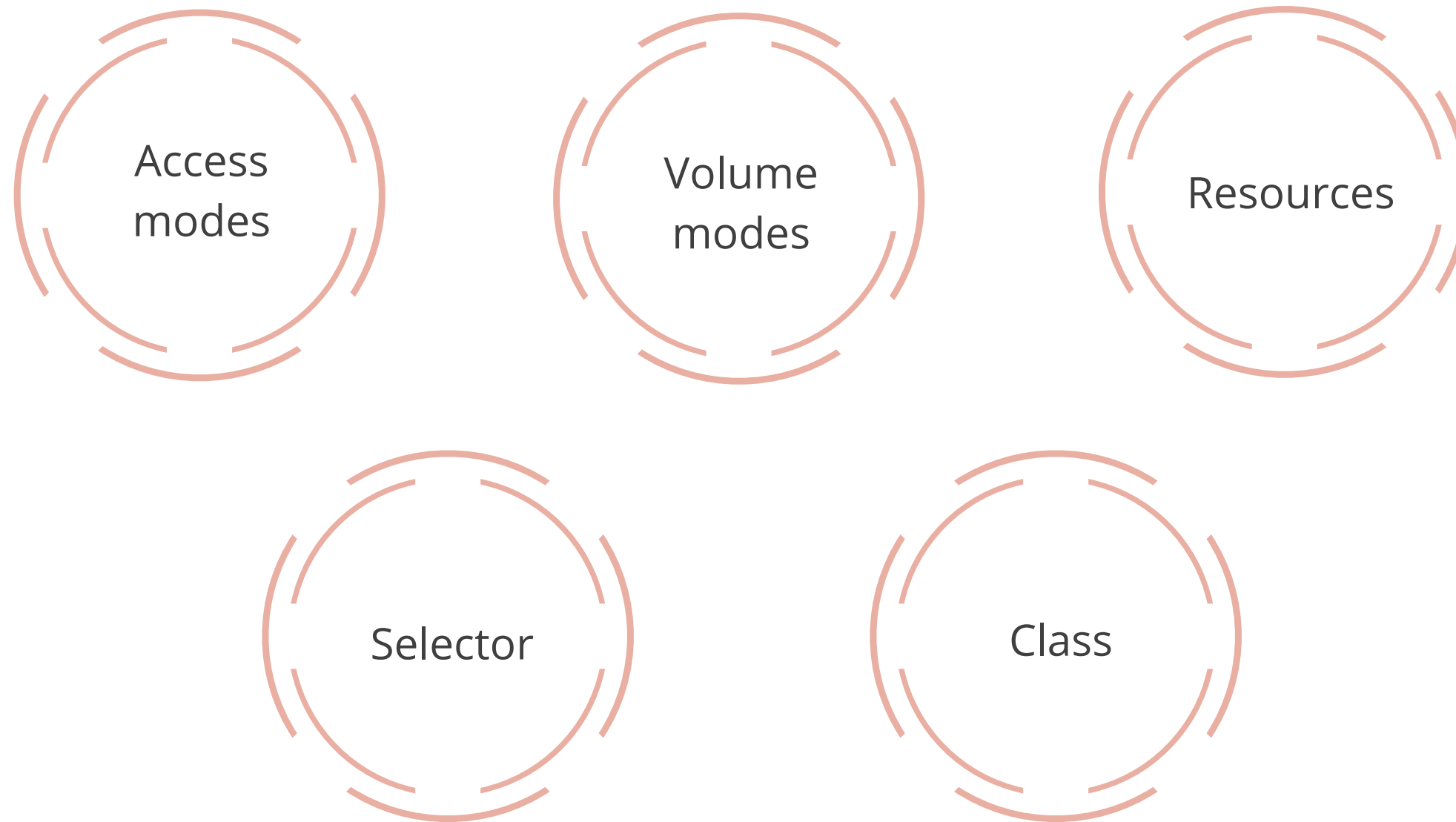
Example:

```
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

These are the specifications available for **PersistentVolumeClaim**:



PersistentVolumeClaim

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

Example:

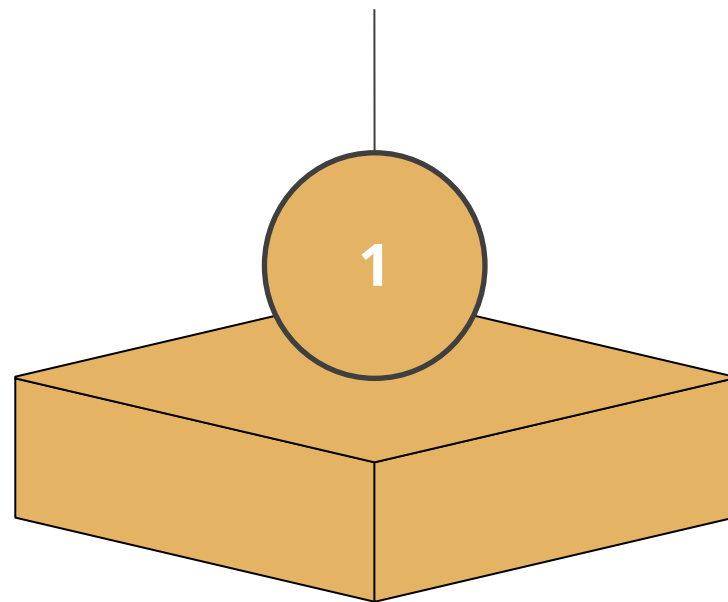
```
  apiVersion: v1
  Kind: PersistentVolumeClaim
  Metadata:
    name: myclaim
  spec:
    accessModes:
      - ReadWriteOnce
    volumeMode: Filesystem
    resources:
      requests:
        storage: 1Gi
    storageClassName : slow
    selector:
      matchLabels:
        release: "stable"
      matchExpressions:
        - {key: environment, operator : In, values: [dev]}
```



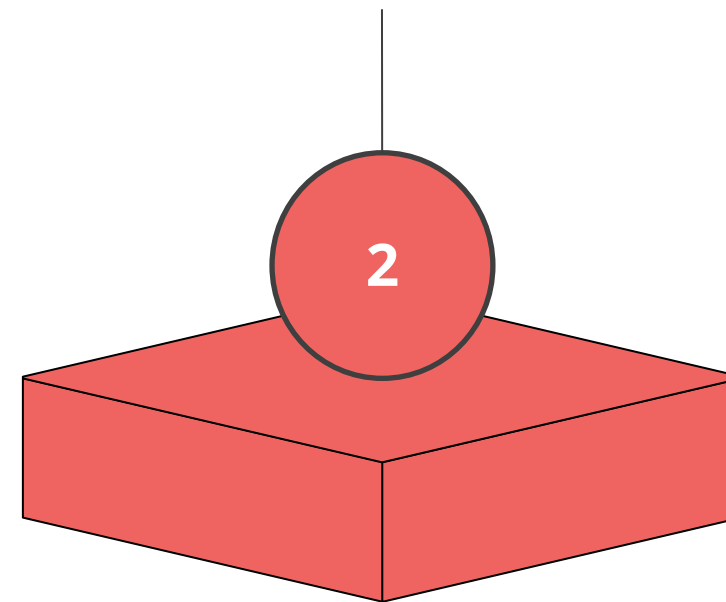
Claim as Volumes

Pods access storage by utilizing the claim as a volume.

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

The following volume plugins support raw block materials:

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:

Example:

```
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

The following box shows how a PersistentVolumeClaim that requests for a Raw Block Volume for an accessMode of ReadWriteOnce can be configured:

Example:

```
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

The following code explains how to add a Raw Block Device path in Container:

Example:

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



Binding Block Volumes

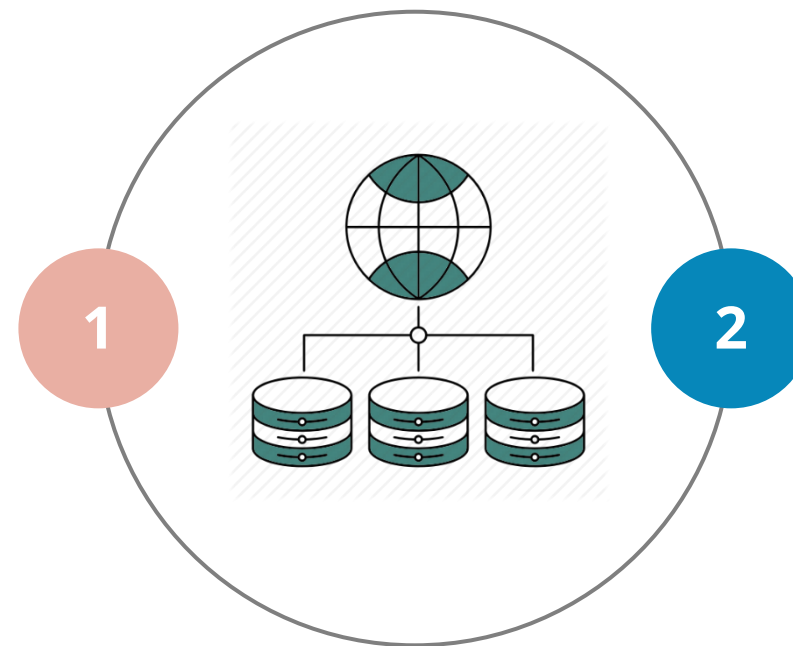
The following table indicates if the volume will be bound or not given the combination:

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

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.

Understanding PersistentVolume (PV) And PersistentVolumeClaim (PVC)



Duration: 20 mins

Problem Statement:

You've been assigned a task to configure a Pod with a PersistentVolume and a PersistentVolumeClaim for storage management.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed:

1. Configuring the NFS kernel server
2. Setting the permissions
3. Configuring the NFS common on client machines
4. Creating a PersistentVolume
5. Creating a PersistentVolumeClaim
6. Creating a Deployment for MySQL



Volume Snapshots

Introduction

VolumeSnapshotContent and **VolumeSnapshot** API resources enable the creation of volume snapshots for users and administrators.

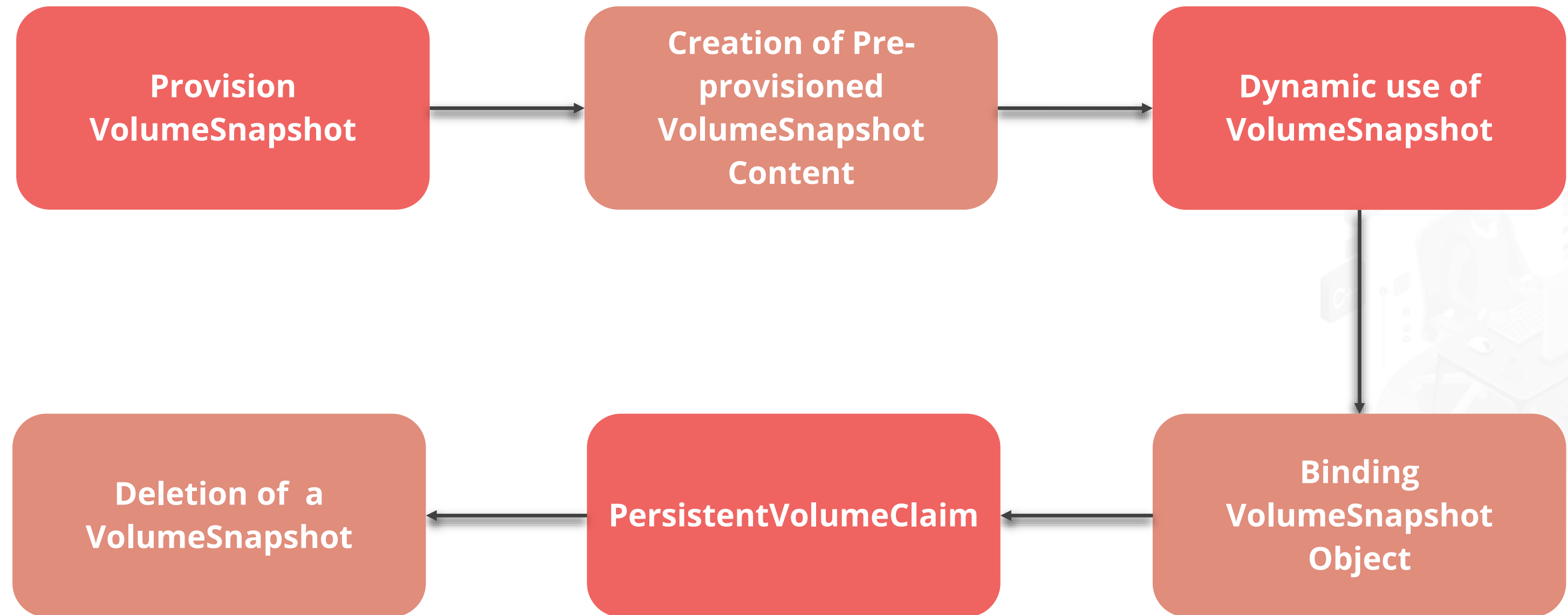
A **VolumeSnapshotContent** is a snapshot originating from a volume in the cluster that has been provisioned by an administrator.

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

The lifecycle of interaction between VolumeSnapshot and VolumeSnapshot Content:



Volume Snapshots

A spec and a status are parts of a VolumeSnapshot.

Example:

```
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.

Example:

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



Provisioning Volumes from Snapshots



The **dataSource** field in the **PersistentVolumeClaim** can be used to provision a new volume that has been pre-populated with data from a snapshot.



VolumeSnapshot Contents

In dynamic provisioning, the snapshot common controller creates **VolumeSnapshotContent** objects.

Example:

```
apiVersion: snapshot.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.

Example:

```
apiVersion:  snapshot.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
    volumeSnapshotRef:
      name:     new-snapshot-test
      namespace: default
```



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 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.

Example:

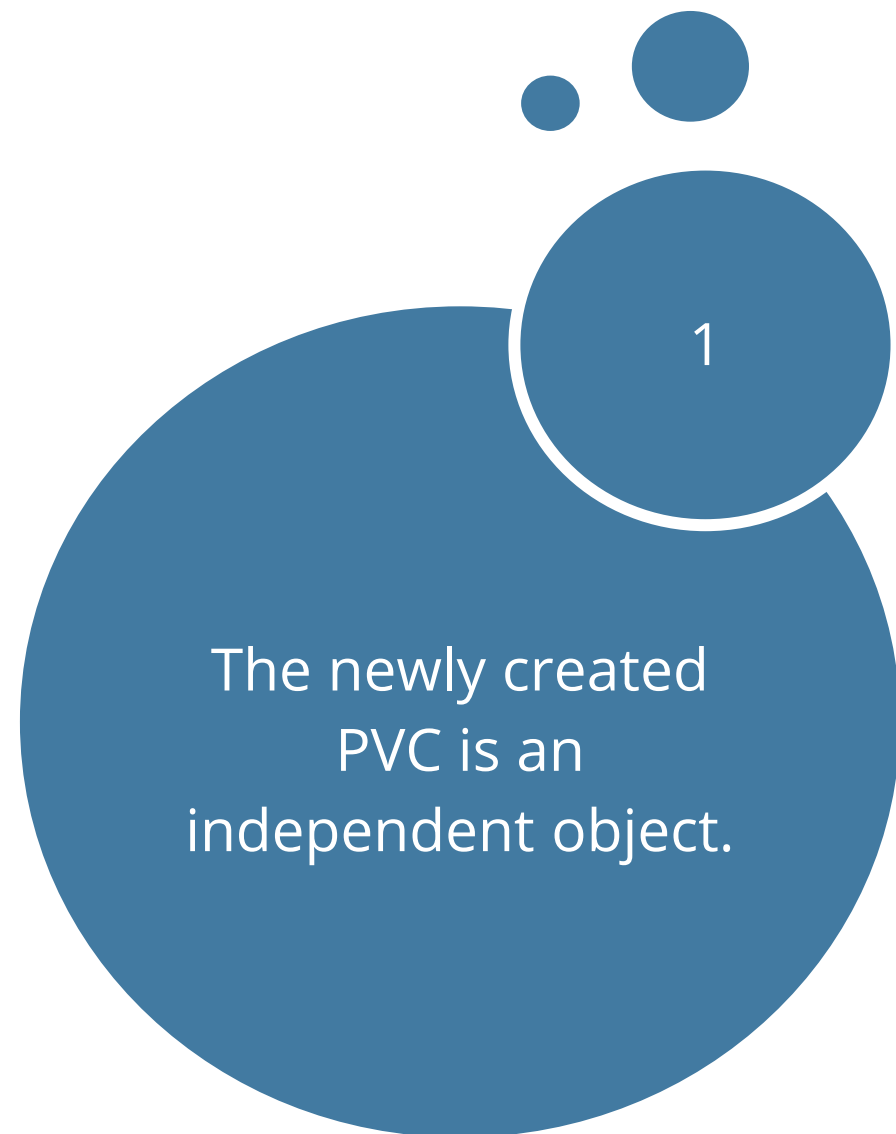
```
apiVersion: v1
Kind: PersistentVolumeClaim
Metadata:
  name: clone-of-pvc-1
  namespace: myns
Spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: cloning

  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 other PVCs.



Storage Classes

Storage Classes

Administrators use **StorageClass** 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.

Example:

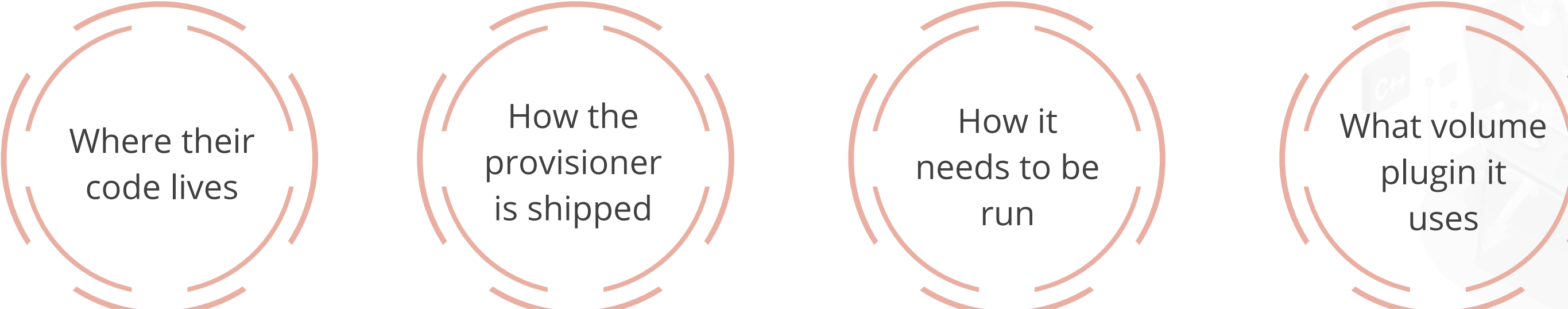
```
apiVersion: storage.k8s.io/v1
Kind: StorageClass
Metadata:
  name: standard
provisioner: kubernetes.io/aws-ebc
parameters:
  type: gp3
reclaimPolicy: Retain
allowVolumeExpansion: true
mountOptions:
  - debug
volumeBindingMode: Immediate
```



Provisioner

Each StorageClass has a provisioner that determines what volume plugin is used for provisioning PVs.

Authors of external provisioners have full discretion over:



Where their
code lives

How the
provisioner
is shipped

How it
needs to be
run

What volume
plugin it
uses

StorageClass Resource: Fields

Reclaim Policy

PersistentVolumes created dynamically by a StorageClass will have the reclaim policy specifics mentioned in the **reclaimPolicy** field of the class. It can be either **Delete** or **Retain**.

Volume Expansion

Configuration of PersistentVolumes can make them expandable.

Mount Options

PersistentVolumes that are dynamically created by a StorageClass will have the mount options specified in the **mountOptions** field of the class.

Volume Binding Mode

The **volumeBindingMode** field controls when volume binding and dynamic provisioning should occur.

Allowed Topologies

When a cluster operator specifies the **WaitForFirstConsumer** volume binding mode, provisioning to specific topologies is not constrained. The following code snippet demonstrates how to restrict the topology of provisioned volumes to specific zones:

Example:

```
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 storage class. A maximum of 512 parameters may be defined for a StorageClass.

The provisioners include the following providers:

- 1 AWS EBS
- 2 GCE PD
- 3 Glusterfs
- 4 OpenStack Cinder

- 5 vSphere
- 6 CSI Provisioner
- 7 vCP Provisioner
- 8 Ceph RBD

Parameters

The provisioners include the following providers:

9 Quo Byte

10 Azure Disk

11 Azure File

12 Portworx Volume

13 ScaleIO

14 StorageOS

15 Local

Volume Snapshot Classes

Introduction

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



VolumeSnapshot Class Resource

Each VolumeSnapshotClass has fields for **driver**, **deletionPolicy**, and **parameters**.

The name and other parameters of a class can be finalized during the creation of VolumeSnapshotClass objects. After creation, updating is not possible.

Example:

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



VolumeSnapshot Class Resource

The **snapshot.storage.kubernetes.io/is-default-class: "true"** annotation can be used to specify the default VolumeSnapshotClass for VolumeSnapshots that do not require a specific class to bind, as shown in the snippet below:

Example:

```
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.

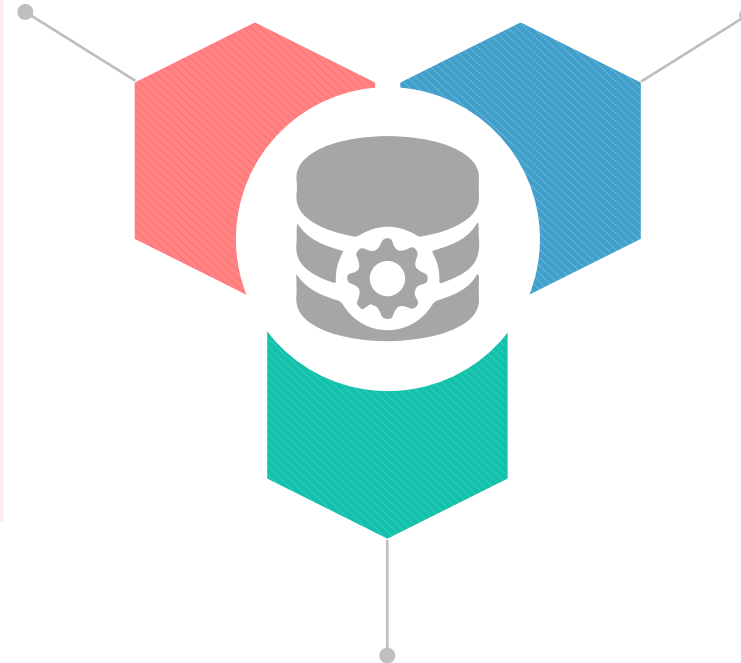


Dynamic Volume Provisioning

Overview

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

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



Automatic storage provision is done upon user request.

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

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:

Example:

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

Example:

```
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:

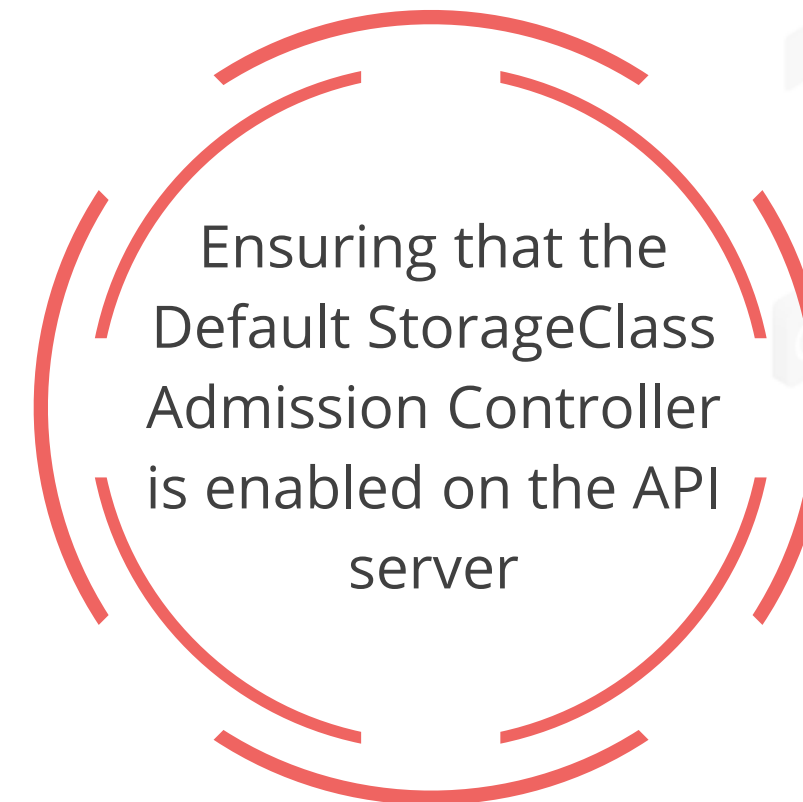
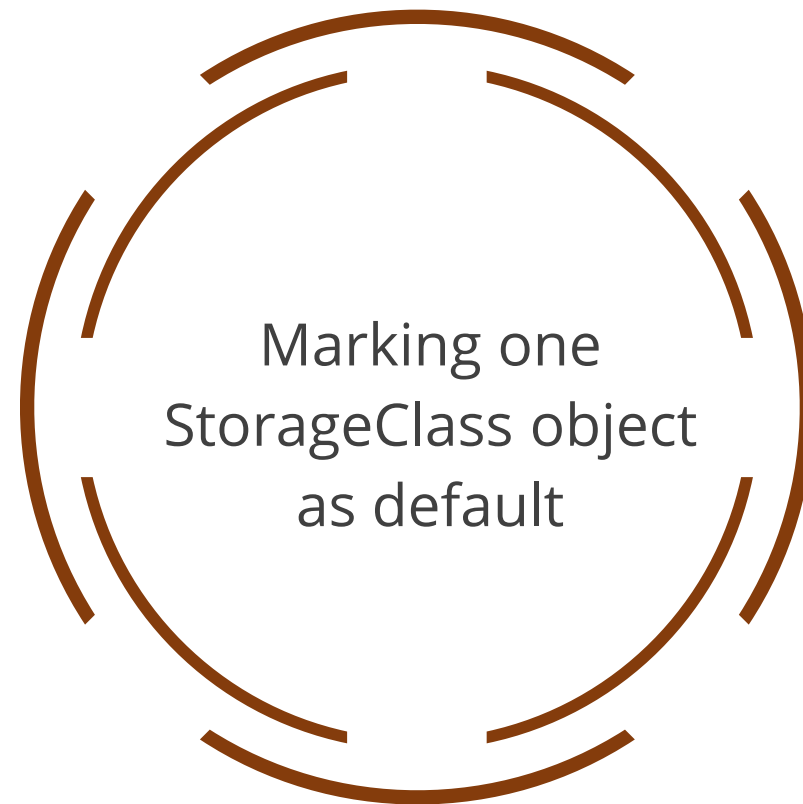
Example:

```
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.



Storage Capacity

Storage Capacity

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

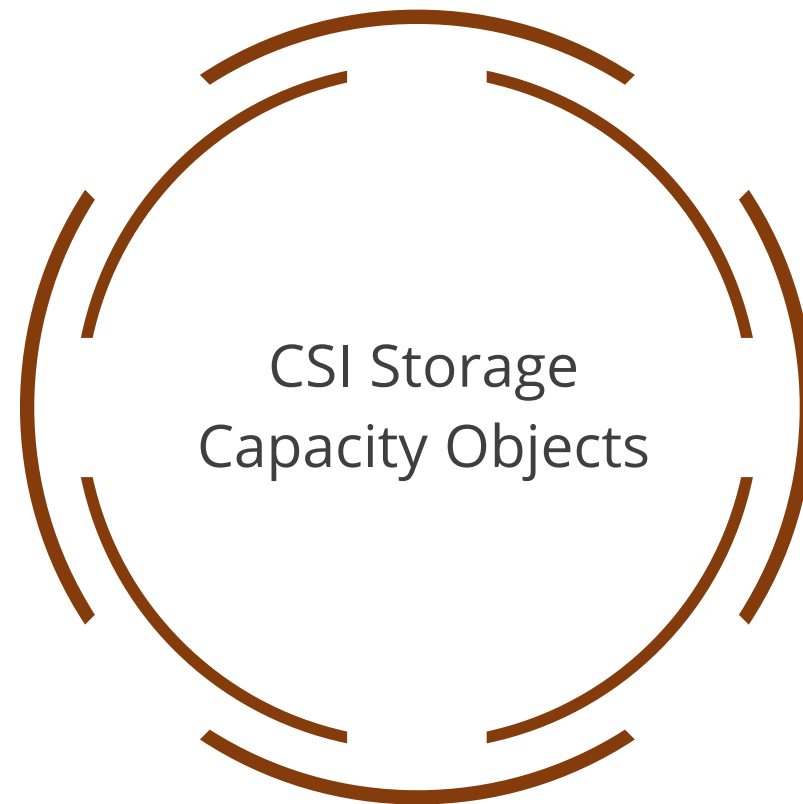


Tracking Storage Capacity is supported for Container Storage Interface (CSI) drivers and should be enabled when a CSI driver is installed.



API

There are two API extensions for this feature:



Criteria Scheduling

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

1

The CSIStorageCapacity feature gate is true

2

A Pod uses a Volume that has not been created

3

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

4

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.

Example:

```
#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"
```



Ephemeral Volumes

Overview

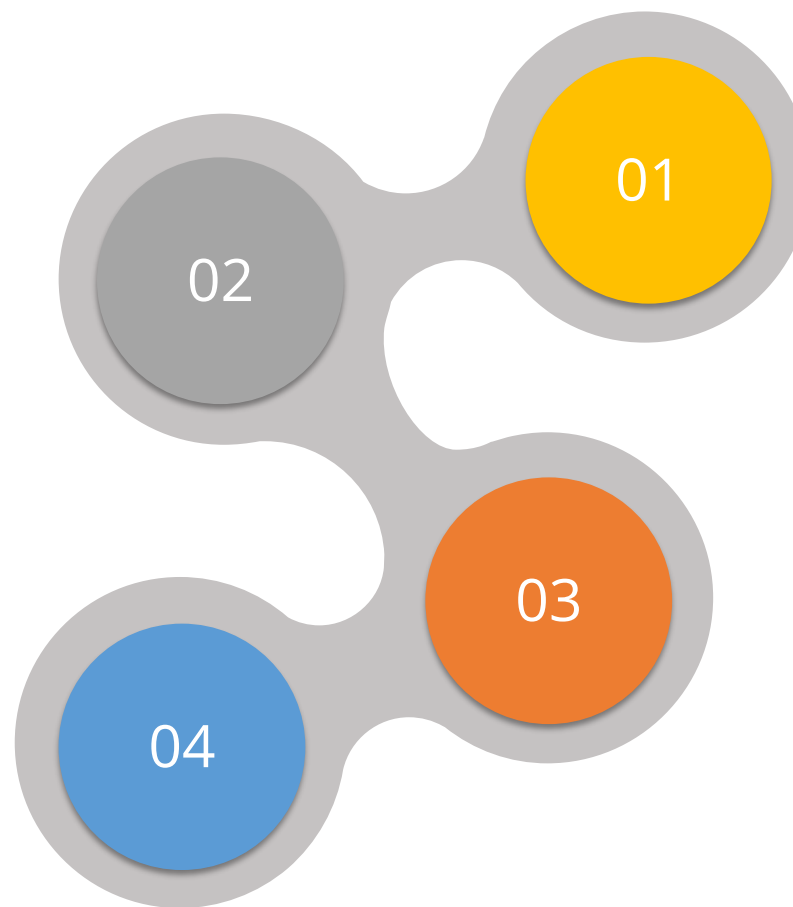
Caching services often have limited memory size. They can shift infrequently-used data into Storage.

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:

Example:

```
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:

Example:

```
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:
```

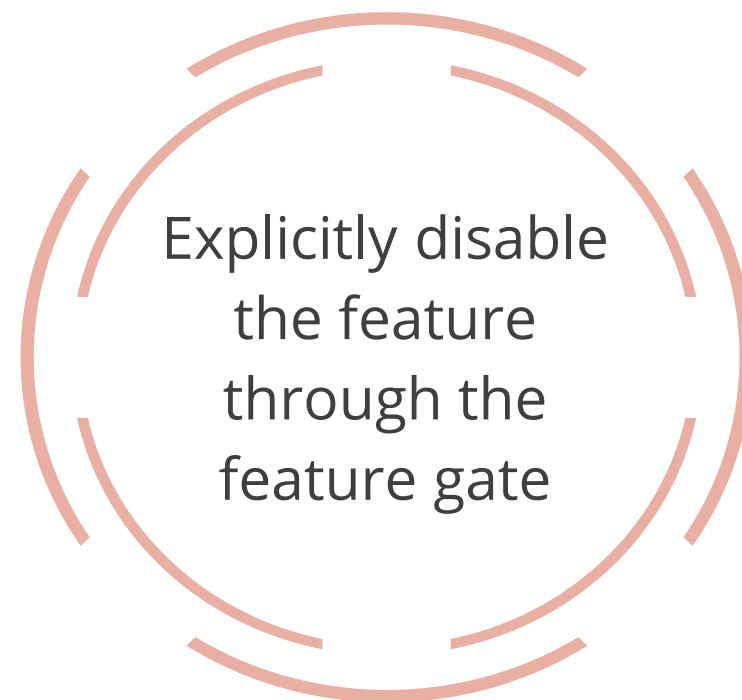
Example:

```
  volumeClaimTemplate:
    Metadata:
      Labels:
    type: my-frontend-volume
    spec:
      inline.storage.kubernetes.io
      accessModes: [
        "ReadWriteOnce" ]
      storageClassName: "scratch-
storage-class"
      Resources:
        requests:
          Storage: 1Gib
```

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.

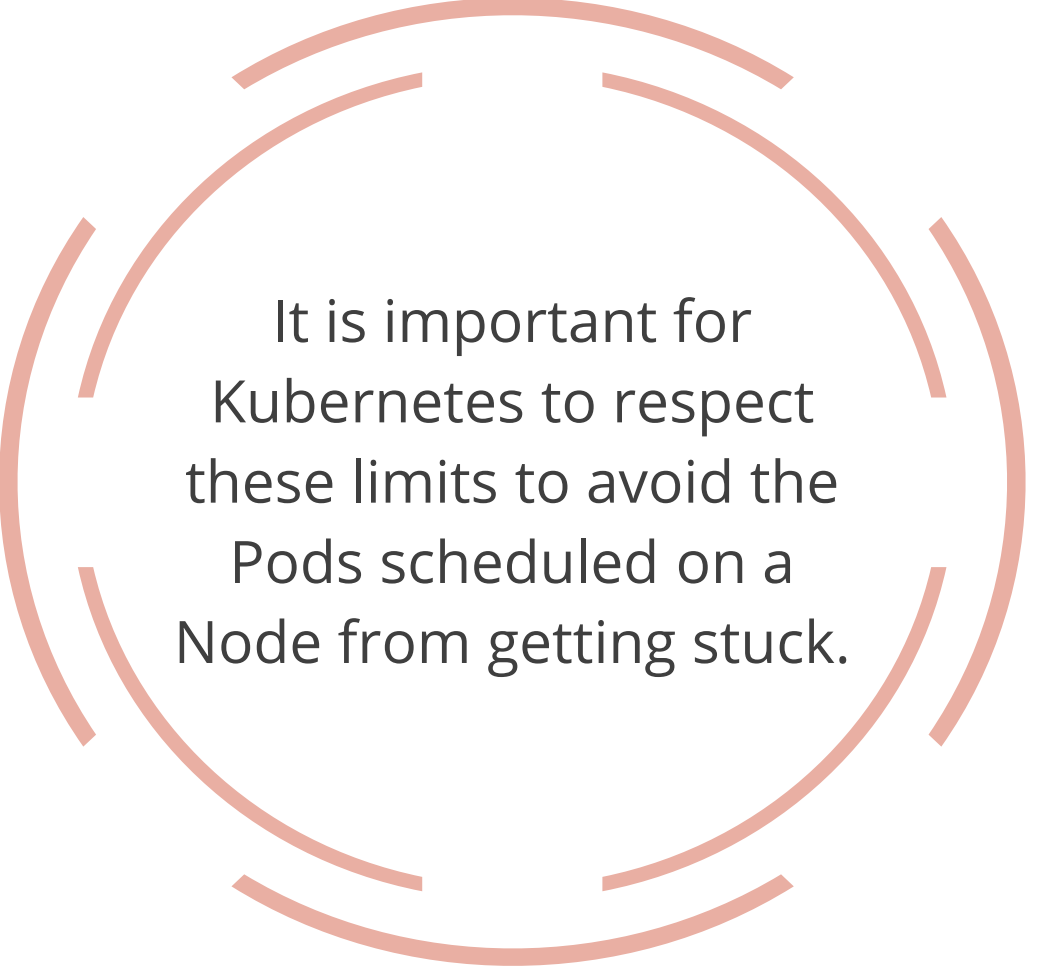
Two choices, if this does not fit the security model, are:



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.

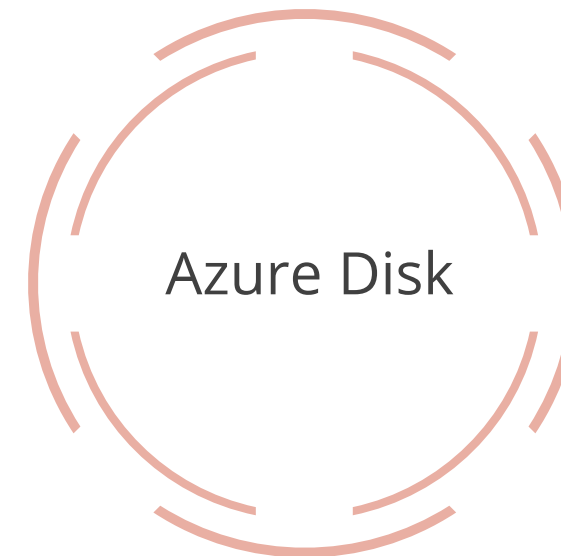
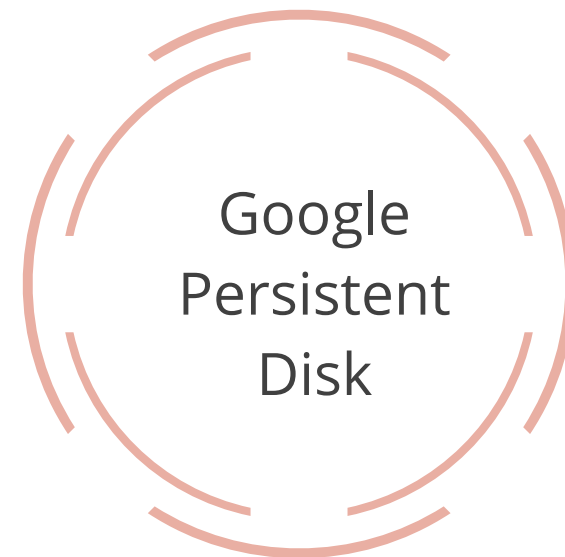


Use caution if you set a limit that is higher than the default limit.



Dynamic Volume Limits

Dynamic Volume Limits are supported for the following:



Configmap as Volume by using Deployment



Duration: 10 Min.

Problem Statement

You've been assigned a task to create a Deployment with Configmap as a volume.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to configmap as Volume by using Deployment:

1. Create configmap and for attach volume to it
2. Verify configmap state
3. Create httpd deployment to attach configmap as volume to it
4. Verify pods and Deployment



Secret as Volume



Duration: 10 Min.

Problem Statement:

You've been asked to create a secret to use it as secret as a volume.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to creating a secret to use it as secret as volume:

1. Creating a secret to use it as secret as volume
2. Creating a Pod to map secret as volume
3. Accessing the Container fetch mounted data by the following command



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.
- The 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 when installing a CSI driver.



Deploy WordPress and MySQL using persistent volume.

Duration: 15 Min.



Project agenda: To deploy WordPress and MySQL using persistent volume.

Description: Your organization is in need of WordPress and MySQL application. WordPress must be deployed using host path and MySQL must be deployed using NFS.

Perform the following:

1. Create a cluster
2. Create an NFS server
3. Deploy NFS client on worker nodes
4. Create a MySQL manifest file and deploy it using NFS-based persistent volume
5. Create a WordPress manifest file and deploy it using host-path-based persistent volume