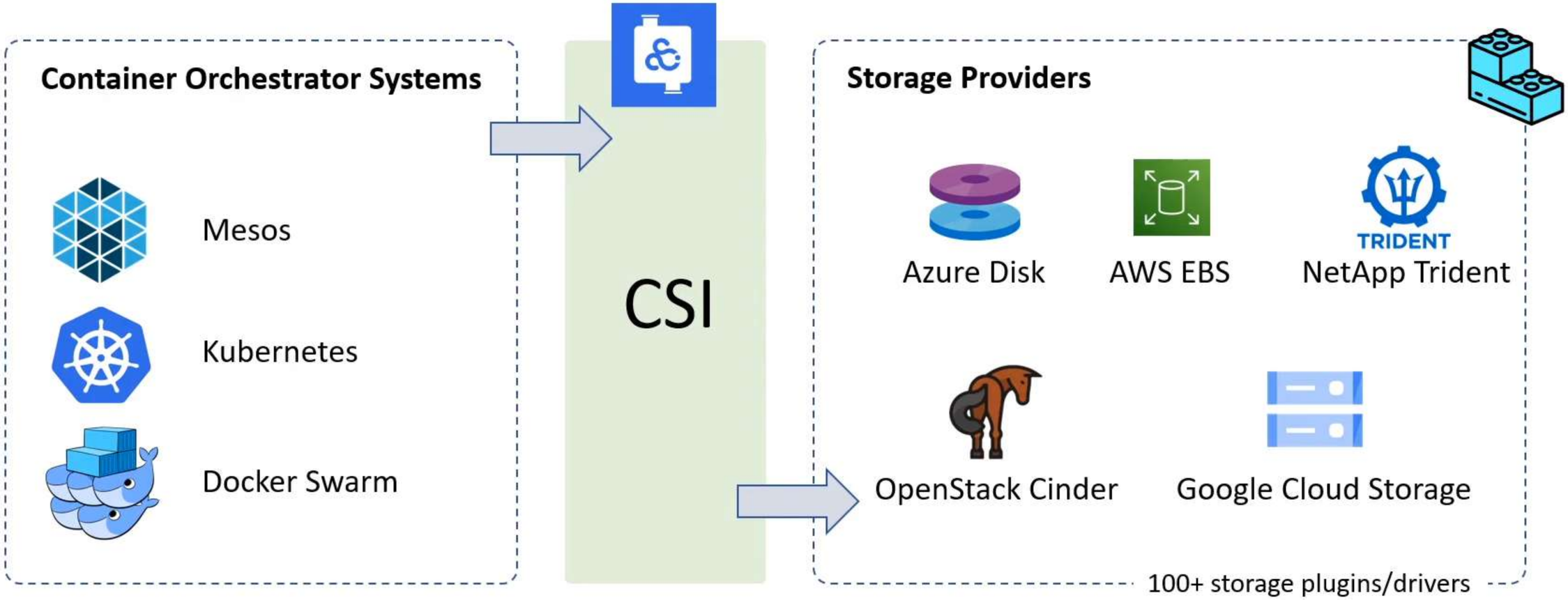
Container Storage Interface (CSI)

Container Storage Interface (CSI) standardizes how
Container Orchestrator Systems (COS) access to various **storage providers**

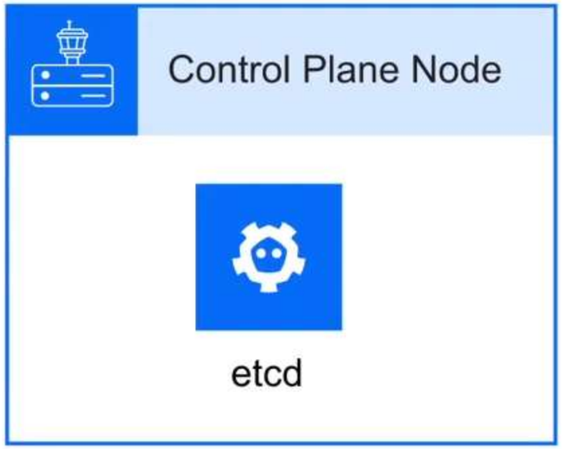


Kubernetes Backing Store and etcd

Components of a Kubernetes cluster include: pods, nodes, control plane, and volumes, and each of these needs a level of protection in case of disaster

Kubernetes resources are stored in an etcd (but could be backed by MariaDB).
Application data is stored in persistent volumes for applications running on the cluster.

etcd is a strongly consistent, distributed key-value store that provides a reliable way to store data that needs to be accessed by a distributed system or cluster of machines.



Things that use etcd:

- Kubernetes Cluster
- CoreDNS
- Rook

MinIO and Rook

Rook turns distributed storage systems into self-managing, self-scaling, self-healing storage services.

It automates the tasks of a storage administrator: deployment, bootstrapping, configuration, provisioning, scaling, upgrading, migration, disaster recovery, monitoring, and resource management.

MinIO offers high-performance, **S3 compatible object storage**.

Native to Kubernetes, MinIO is the only object storage suite available on every public cloud, every Kubernetes distribution, the private cloud and the edge.

MinIO is software-defined and is 100% open source under GNU AGPL v3.

Volumes

Kubernetes supports many types of volumes.

A Pod can use any number of volume types simultaneously

Persistent Volumes
Attaching an external storage to a pod.
The data will persist even if the pod is terminated.

Ephemeral Volumes
A volume that's only exists as long as the pod exists.
Intended for temporary data storage.

Projected Volumes
maps several existing volume sources into the same directory.

Volume Snapshot
Archiving a volume configuration and its data for rollbacks or backup

Types of **Volumes Supported**:

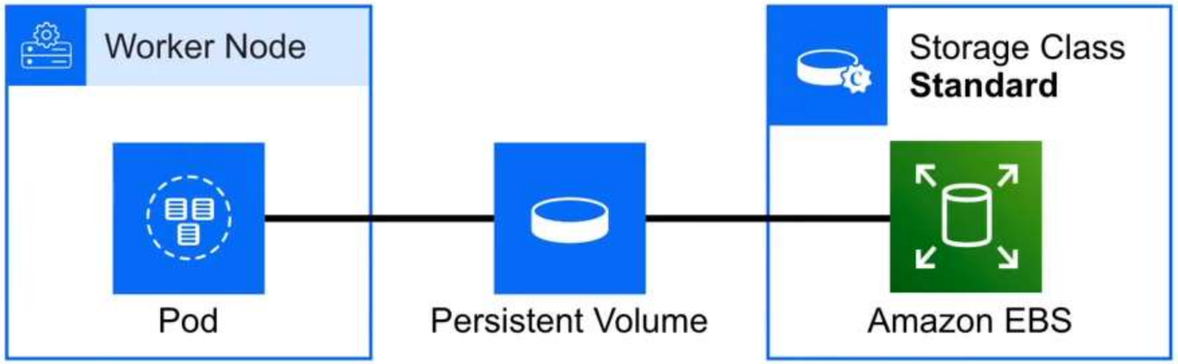
- awsElasticBlockStore
- azureDisk
- azureFile
- cephfs
- cinder
- configMap
- downwardAPI
- emptyDir
- fc (fibre channel)
- gcePersistentDisk
- glusterfs
- hostPath
- iscsi
- local
- nfs
- persistentVolumeClaim
- portworxVolume
- projected
- rbd
- secret
- vsphereVolume

Persistent Volume (PV)

A piece of storage in the cluster that has been provisioned by an administrator or dynamically provisioned using Storage Classes

PV is similar to node in that is a cluster resource.

PV does not reside within a node or pod.



This diagram is **technically not accurate*

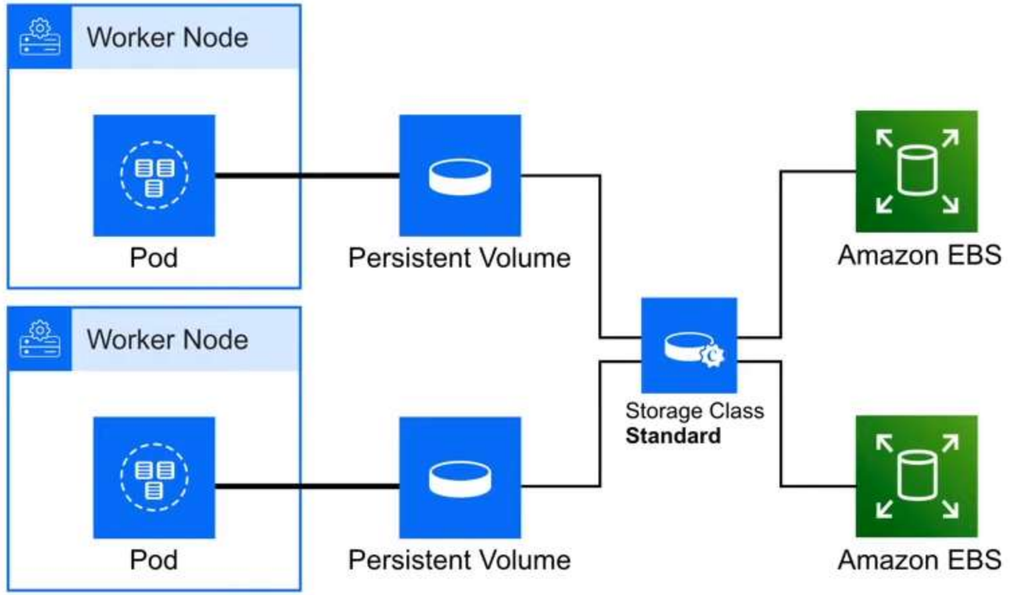
PVs are volume plugins like Volumes, but have a lifecycle independent of any individual Pod that uses the PV

This API object captures the details of the implementation of the storage, be that NFS, iSCSI, or a cloud-provider-specific storage system

mounting pv directly to a pod is not allowed and is against the Kubernetes design principles. It would cause tight coupling below the pod volume and the underlying storage.
Persistent Volume Claims ensures decoupling

Storage Classes

A storage class is way of defining a class of storage that is backed by a provisioner.




```yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: standard
provisioner: kubernetes.io/aws-ebs
parameters:
  type: gp2
reclaimPolicy: Retain
allowVolumeExpansion: true
mountOptions:
  - debug
volumeBindingMode: Immediate
```

Storage Class can be used by many Persistent Volumes.

Storage class contains:

- Provisioner – who is storage? Eg. Amazon EBS
- Parameters – what type of storage of Amazon EBS to use
- reclaimPolicy — If the pod is gone does the volume remain?

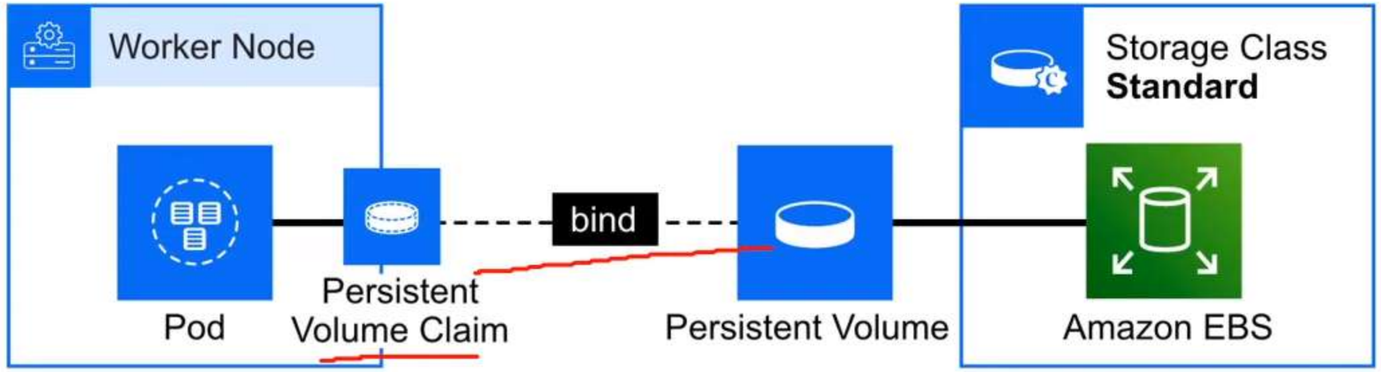
Persistent Volume Claim (PVC)

A Persistent Volume Claim (PVC) is used to decouple Persistent Volumes from pods. A PVC asked for a type of storage, and if a PV that meets the criteria is matched, the a PV is Claimed and Bound.

PVCs similar to a Pod requesting resources from a Node

- Pods consume node resources
  - PVCs consume PV resources
- Pods can request specific levels of resources (CPU and Memory)
  - PVCs can request specific size and access modes (e.g., they can be mounted ReadWriteOnce, ReadOnlyMany or ReadWriteMany, see AccessModes).

```yaml
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]}
```

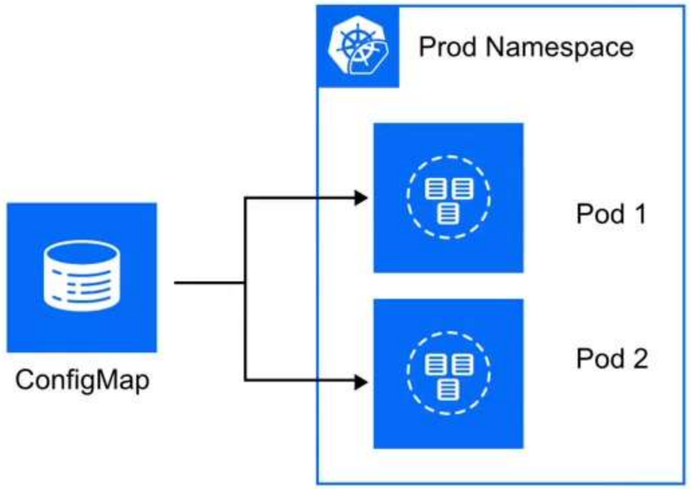


ConfigMaps

A ConfigMap is an API object used to store **non-confidential data** in key-value pairs for pods

Pods can consume ConfigMaps as:

- environment variables
- command-line arguments
- configuration files in a volume




```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: game-demo
data:
  # property-like keys; each key maps to a simple value
  player_initial_lives: "3"
  ui_properties_file_name: "user-interface.properties"
  # file-like keys
  game.properties: |
    enemy.types=aliens,monsters
    player.maximum-lives=5
  user-interface.properties: |
    color.good=purple
    color.bad=yellow
    allow.textmode=true
```

A ConfigMap allows you to decouple environment-specific configuration from your container images, so that your applications are easily portable.