kubernetes

Managing Storage

Kubernetes

Course Objectives

In this module, you will learn:

- Volumes
- Persistent Volumes
- Persistent Volume Claims

kubernetes

Volumes, PV & PVC

Managing Storage

Kubernetes Volumes solves below problems

- On-disk files in a container are ephemeral and they are lost when a container crashes. when a container crashes, the kubelet restarts the container but with a clean state.
- A second problem occurs when sharing files between containers running together in a Pod.

Introduction to Volumes

- A Kubernetes volume is essentially a directory accessible to all containers running in a pod.
- In contrast to the container-local filesystem, the data in volumes is preserved across container restarts.
- When a pod ceases to exist, the volume is destroyed.

Volumes Types

- The medium backing a volume and its contents are determined by the volume type:
  - node-local types such as `emptyDir` or `hostPath`
  - file-sharing types such as `nfs`
  - cloud provider-specific types like `awsElasticBlockStore`, `azureDisk`, or `gcePersistentDisk`
  - distributed file system types, for example `glusterfs` or `cephfs`
  - special-purpose types like `secret`, `gitRepo`

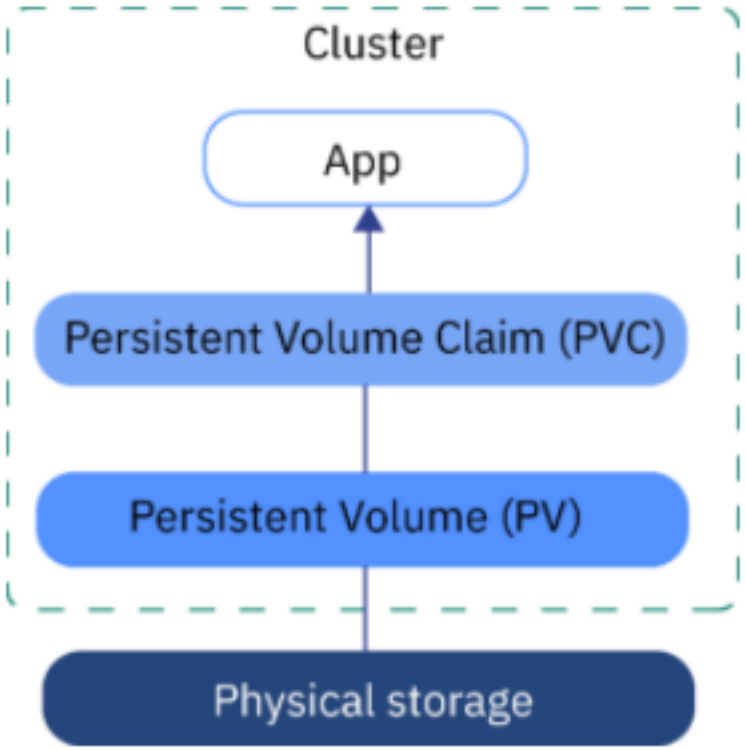
emptyDir

- An `emptyDir` volume is first created when a Pod is assigned to a node, and exists as long as that Pod is running on that node.
- As the name says, the emptyDir volume is initially empty. All containers in the Pod can read and write the same files in the emptyDir volume, though that volume can be mounted at the same or different paths in each container.
- When a Pod is removed from a node for any reason, the data in the emptyDir is deleted permanently.
- **Note:** A container crashing does *not* remove a Pod from a node. The data in an emptyDir volume is safe across container crashes.
- Some uses for an emptyDir are:
  - scratch space, such as for a disk-based merge sort
  - checkpointing a long computation for recovery from crashes

Kubernetes Volumes vs Persistent Volumes

- A **Kubernetes volume** exists only while the containing pod exists.
  - Once the pod is deleted, the associated volume is also deleted.
  - Kubernetes volumes are useful for storing temporary data that does not need to exist outside of the pod's lifecycle.
- **Persistent volumes** remain available outside of the pod lifecycle.
  - This means that the volume will remain even after the pod is deleted.
  - It is available to claim by another pod if required, and the data is retained.

PV & PVC


Cluster
App
Persistent Volume Claim (PVC)
Persistent Volume (PV)
Physical storage

Persistent Volumes

- PVs are defined by a PersistentVolume API object, which represents a piece of **existing networked storage** in the cluster that has been provisioned by an administrator.
- It is a resource in the cluster just like a node is a cluster resource.
- PVs are volume plug-ins like Volumes, but have a lifecycle independent of any individual pod that uses the PV.
- PV objects capture the details of the implementation of the storage, be that NFS, iSCSI, or a cloud-provider-specific storage system.

Persistent Volume Claims

- PVCs are defined by a PersistentVolumeClaim API object, which represents a request for storage by a developer.
- It is similar to a pod in that pods consume node resources and PVCs consume PV resources.
- For example, pods can request specific levels of resources (e.g., CPU and memory), while PVCs can request specific storage capacity and access modes (e.g, they can be mounted once read/write or many times read-only).

Lifecycle of a Volume and Claim

- PVs are resources in the cluster.
- PVCs are requests for those resources and also act as claim checks to the resource.
- The interaction between PVs and PVCs have the following lifecycle.

Stage 1 - Provisioning

- A cluster administrator creates some number of PVs.
- They carry the details of the real storage that is available for use by cluster users.
- They exist in the API and are available for consumption.

Lifecycle of a Volume and Claim

Stage 2 - Binding

- A user creates a PersistentVolumeClaim with a specific amount of storage requested and with certain access modes.
- A control loop in the master watches for new PVCs, finds a matching PV (if possible), and binds them together.
- The user will always get at least what they asked for, but the volume may be in excess of what was requested. To minimize the excess, Container Platform binds to the smallest PV that matches all other criteria.
- Claims remain unbound indefinitely if a matching volume does not exist.
- Claims are bound as matching volumes become available.
- For example, a cluster provisioned with many 50Gi volumes would not match a PVC requesting 100Gi. The PVC can be bound when a 100Gi PV is added to the cluster.

Lifecycle of a Volume and Claim

Stage 3 - Using

- Pods use claims as volumes.
- The cluster inspects the claim to find the bound volume and mounts that volume for a pod.
- For those volumes that support multiple access modes, the user specifies which mode is desired when using their claim as a volume in a pod.
- Once a user has a claim and that claim is bound, the bound PV belongs to the user for as long as they need it.
- Users schedule pods and access their claimed PVs by including a PVC in their pod's volumes block.

Lifecycle of a Volume and Claim

Stage 4 - Releasing

- When a user is done with a volume, they can delete the PVC object from the API which allows reclamation of the resource.
- The volume is considered "released" when the claim is deleted, but it is not yet available for another claim.
- The previous claimant’s data remains on the volume which must be handled according to policy.

Lifecycle of a Volume and Claim

Stage 5 - Reclaiming

- The reclaim policy of a PersistentVolume tells the cluster what to do with the volume after it is released.
- Currently, volumes can either be *Retain* or *Recycle*.
- Retain allows for manual reclamation of the resource.
- For those volume plug-ins that support it, recycling performs a basic scrub on the volume (e.g., rm -rf /<volume>/*) and makes it available again for a new claim.

Requesting Storage

- You can request storage by creating **PersistentVolumeClaim** objects in your projects:

```yaml
apiVersion: "v1"
kind: "PersistentVolumeClaim"
metadata:
  name: "claim1"
spec:
  accessModes:
    - "ReadWriteOnce"
  resources:
    requests:
      storage: "5Gi"
  volumeName: "pv0001"
```

Volume & Claim Binding

- A **PersistentVolume** is a specific resource.
- A **PersistentVolumeClaim** is a request for a resource with specific attributes, such as storage size.
- In between the two is a process that matches a claim to an available volume and binds them together.
- This allows the claim to be used as a volume in a pod.
- **Ex:** OpenShift Enterprise finds the volume backing the claim and mounts it into the pod.

Claims as Volumes in Pods

- A **PersistentVolumeClaim** is used by a pod as a volume.
- Ex: OpenShift Enterprise finds the claim with the given name in the same namespace as the pod, then uses the claim to find the corresponding volume to mount.

```
apiVersion: "v1"
kind: "Pod"
metadata:
  name: "mypod"
  labels:
    name: "frontendhttp"
spec:
  containers:
    -
      name: "myfrontend"
      image: "nginx"
      ports:
        -
          containerPort: 80
          name: "http-server"
      volumeMounts:
        -
          mountPath: "/var/www/html"
          name: "pvol"
  volumes:
    -
      name: "pvol"
      persistentVolumeClaim:
        claimName: "claim1"
```

kubernetes

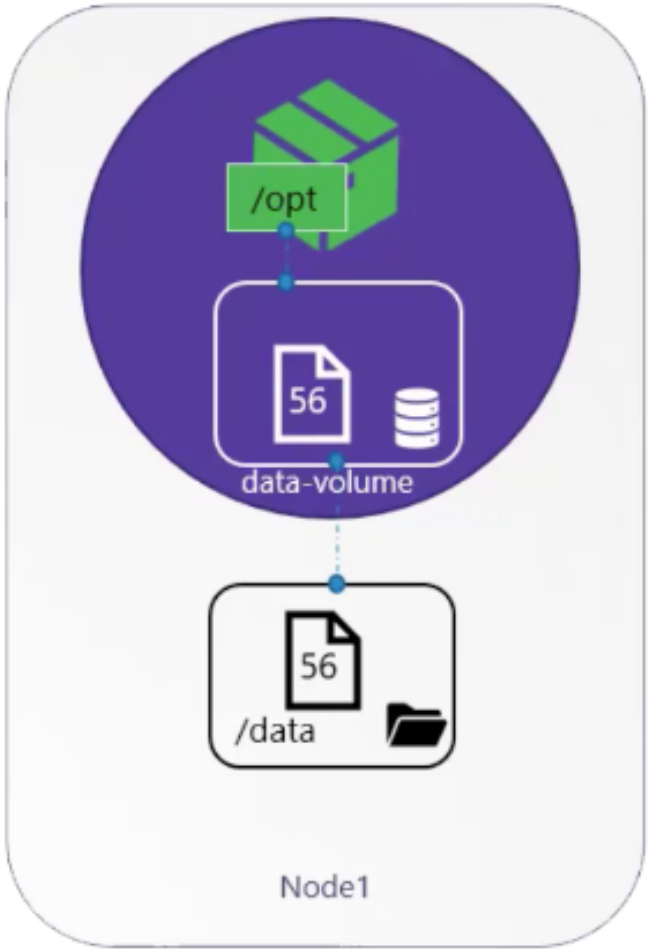
Volumes

Managing Storage

Volumes & Mounts

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: random-number-generator
spec:
  containers:
  - image: alpine
    name: alpine
    command: ["/bin/sh","-c"]
    args: ["shuf -i 0-100 -n 1 >> /opt/number.out;"]
    volumeMounts:
    - mountPath: /opt
      name: data-volume

  volumes:
  - name: data-volume
    hostPath:
      path: /data
      type: Directory
```



Volume Storage Options

```
volumes:
- name: data-volume
  hostPath:
    path: /data
    type: Directory
```

Volume Storage Options

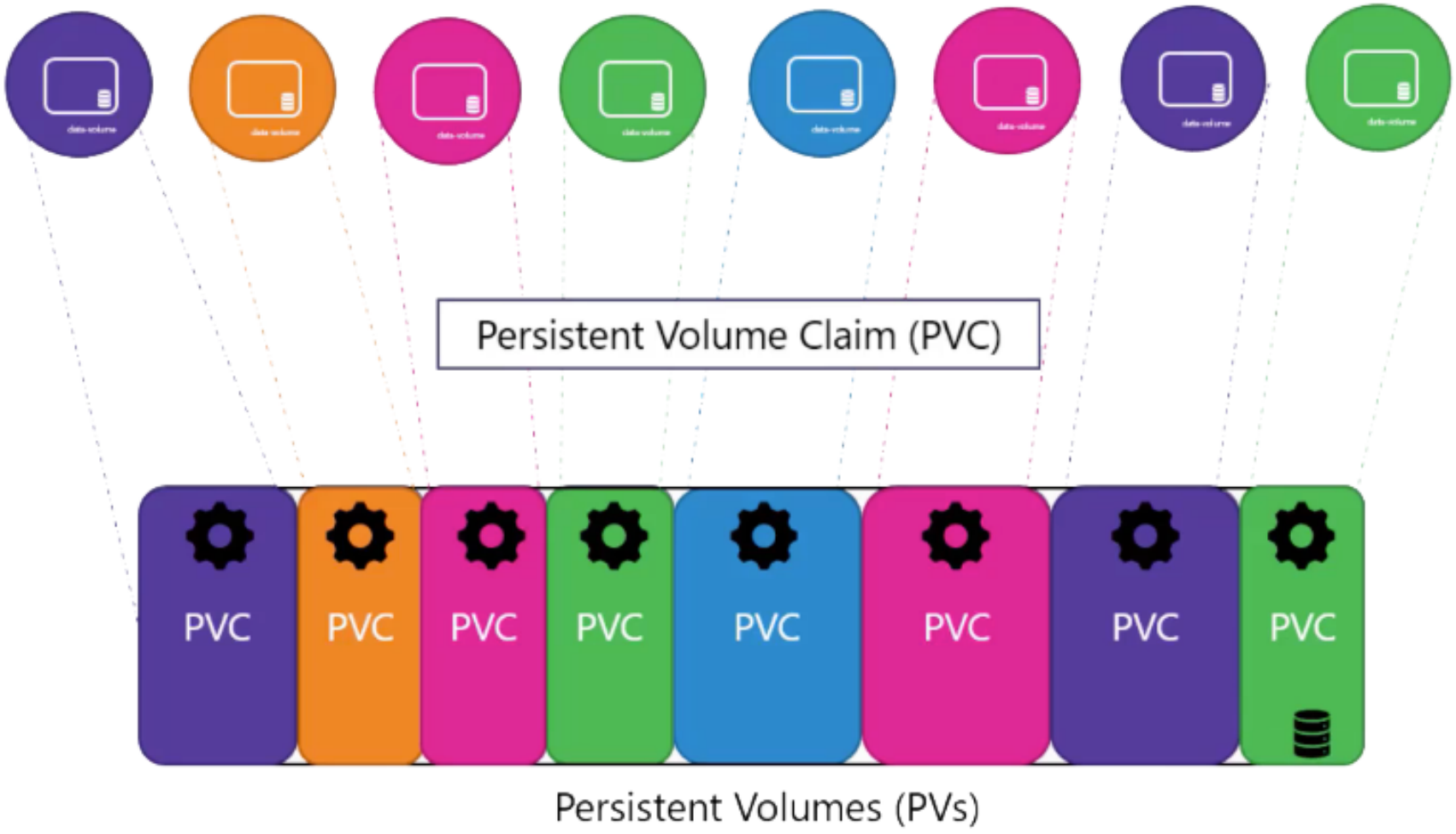
```
volumes:
- name: data-volume
  awsElasticBlockStore:
    volumeID: <volume-id>
    fsType: ext4
```

kubernetes

Persistent Volumes

Managing Storage

Pods using Persistent Volumes



Creating a Persistent Volume

pv-definition.yaml

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
   name: pv-vol1
spec:
  accessModes:
      - ReadWriteOnce
```

- ReadOnlyMany
- ReadWriteOnce
- ReadWriteMany

Persistent Volume (PV)

Persistent Volume Capacity

pv-definition.yaml

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
   name: pv-vol1
spec:
  accessModes:
      - ReadWriteOnce
  capacity:
      storage: 1Gi
  awsElasticBlockStore:
    volumeID: <volume-id>
    fsType: ext4
```

```
kubectl create -f pv-definition.yaml
```

Persistent Volume (PV)

```
kubectl get persistentvolume
```

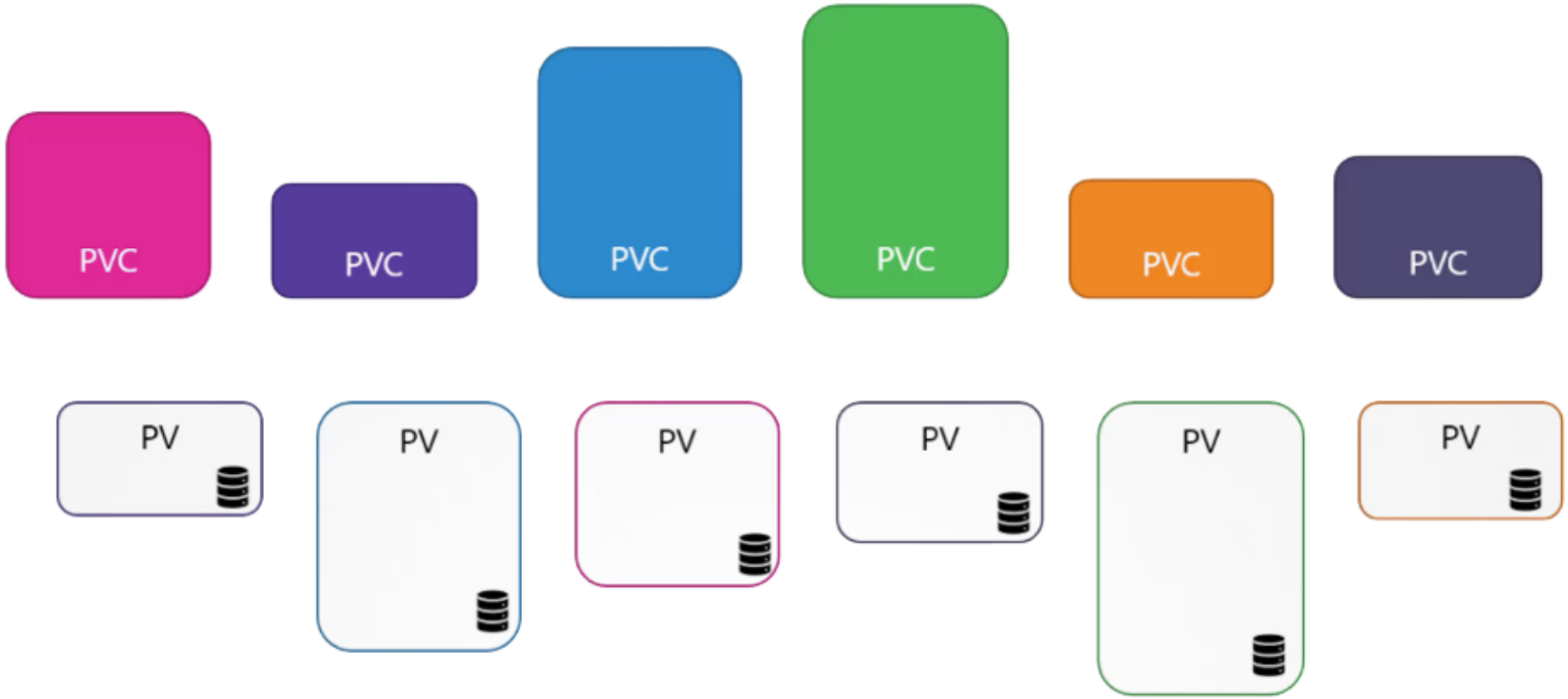
NAME	CAPACITY	ACCESS MODES	RECLAIM POLICY	STATUS	CLAIM	STORAGECLASS	REASON	AGE
pv-vol1	1Gi	RWO	Retain	Available				3m

kubernetes

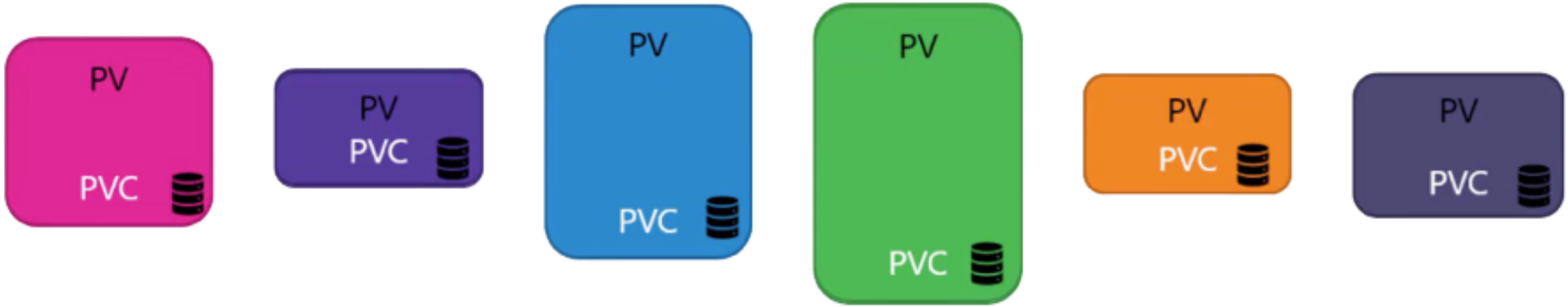
Persistent Volume Claims

Managing Storage

Making Storage available to a node using PVC



Mapping PV to PVC



Sufficient Capacity

Access Modes

Volume Modes

Storage Class

Using labels & Selectors to use specific Volume

Sufficient Capacity | Access Modes | Volume Modes | Storage Class | Selector

Creating Persistent Volume Claim

pvc-definition.yaml

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: myclaim
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 500Mi
```

```
kubectl create -f pvc-definition.yaml
```

```
kubectl get persistentvolumeclaim
NAME        STATUS    VOLUME    CAPACITY   ACCESS MODES
myclaim     Pending
```

Commands

```
kubectl get persistentvolumeclaim
```

```
NAME        STATUS    VOLUME    CAPACITY   ACCESS MODES   STORAGECLASS   AGE
myclaim     Bound     pv-vol1   1Gi        RWO                           43m
```

```
kubectl delete persistentvolumeclaim myclaim
```

```
persistentvolumeclaim "myclaim" deleted
```