Kubernetes Volumes

김영한

참조: TechWorld with Nana

기초용어

- Drive: storage device
- Volume: Logical drive, single accessible storage area with a single file system

  예) Windows c:,d:,….

◆ Mount Volume --- 특정위치에 연결시키는것

본 강좌에서는 Container 에서 Mount Volume 하는 방법을 학습함

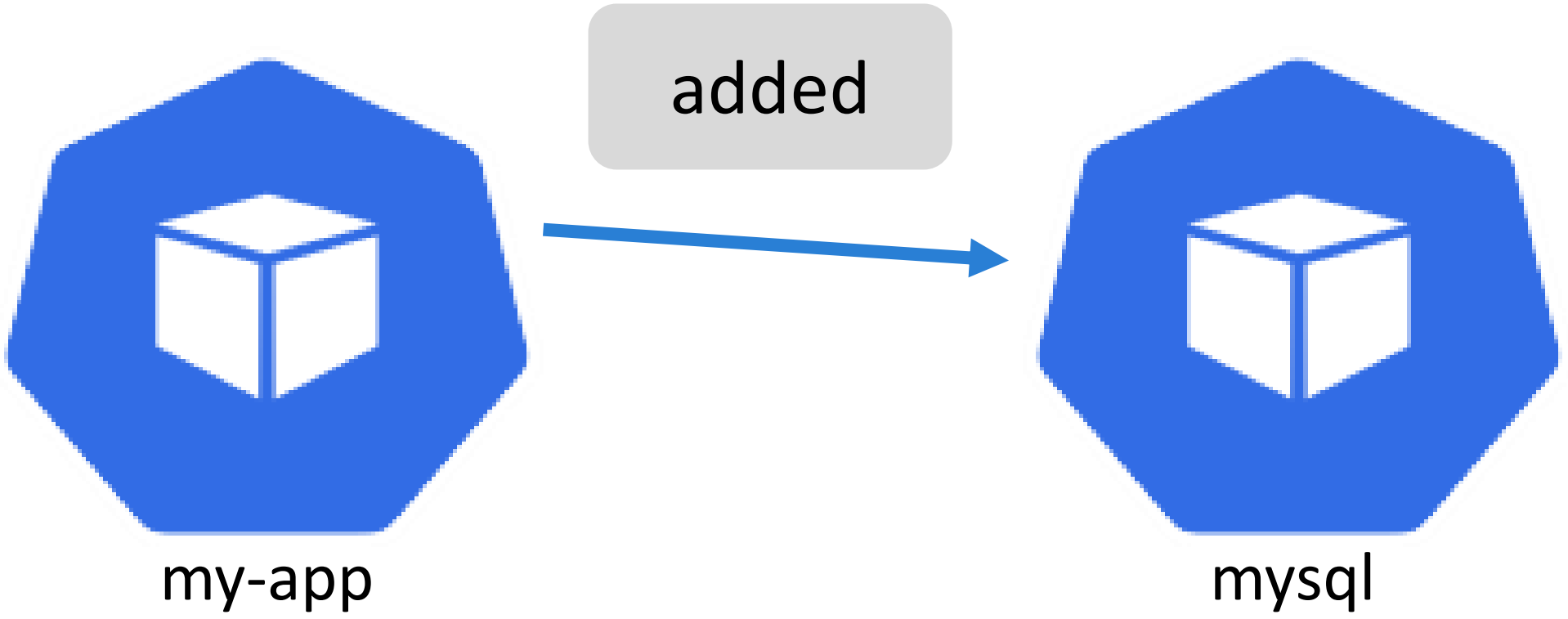
How to persist data in Kubernetes using volumes?

Persistent Volume

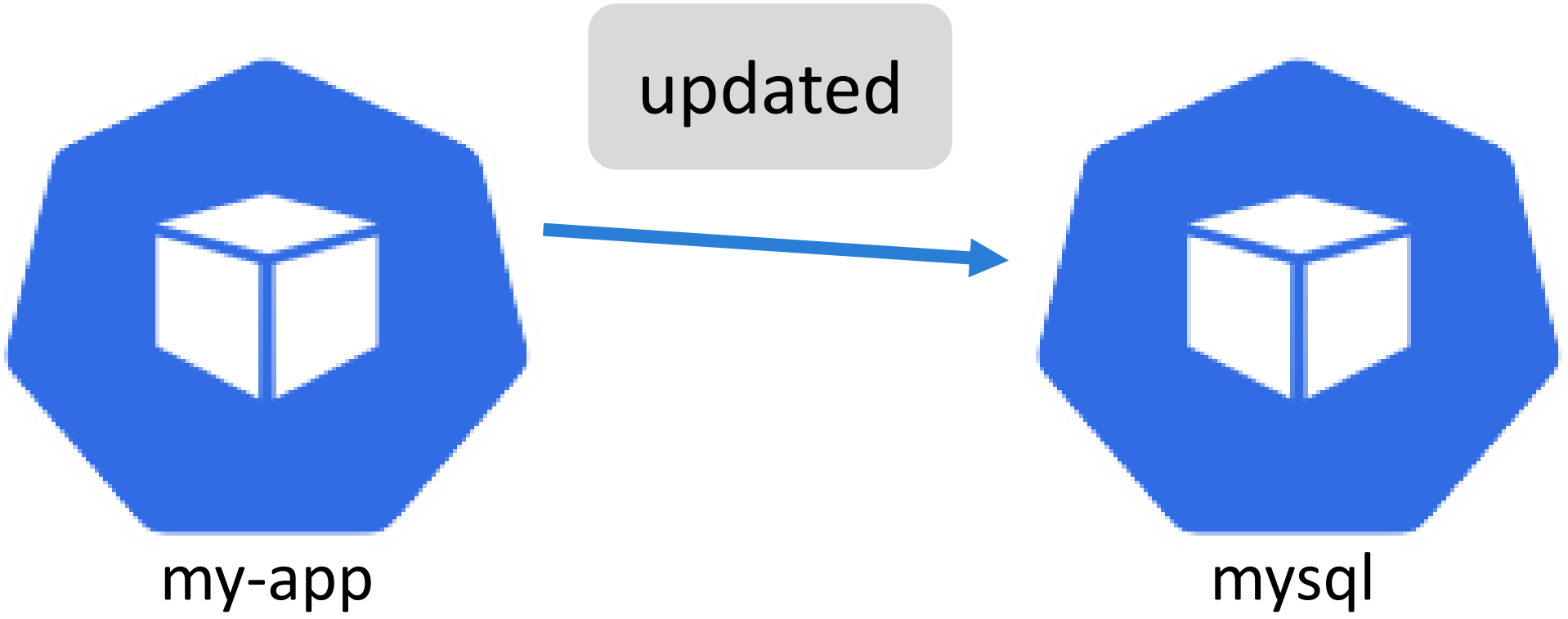
Persistent Volume Claim

Storage Class

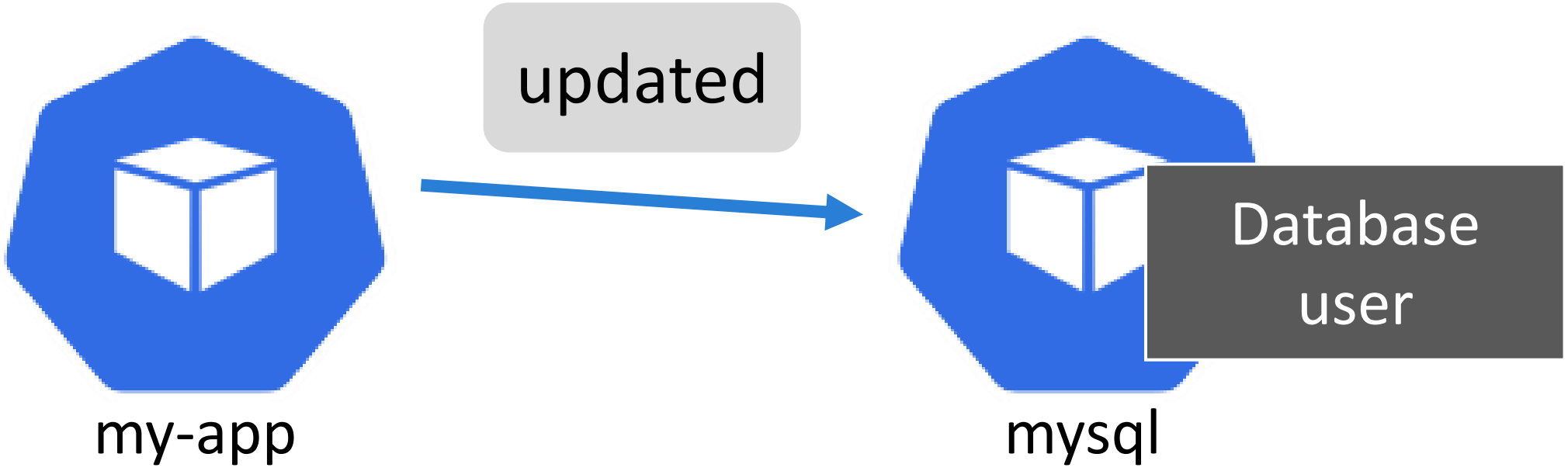
The need for Volumes



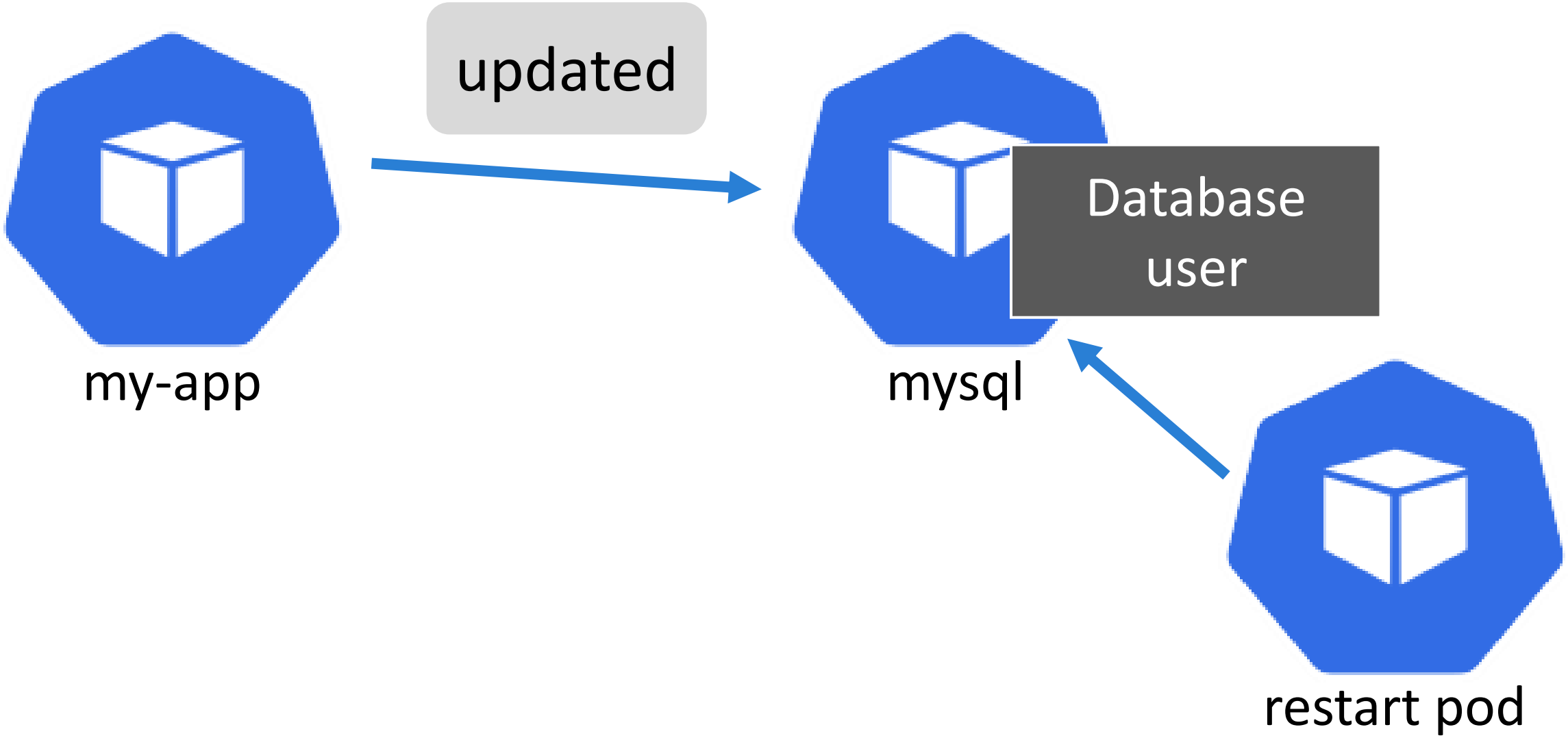
The need for Volumes



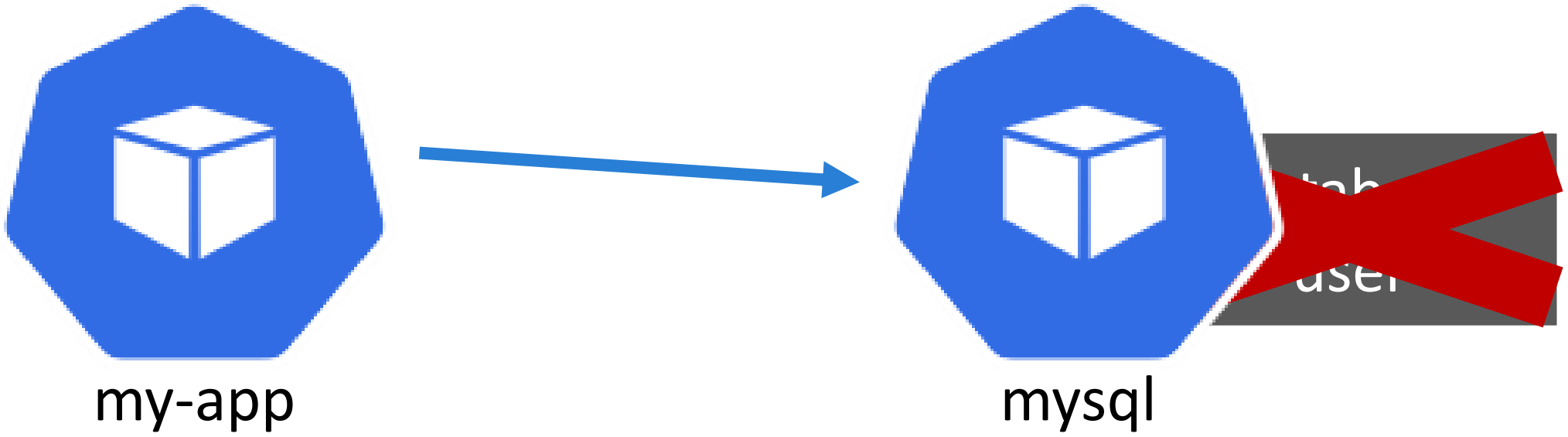
The need for Volumes



The need for Volumes

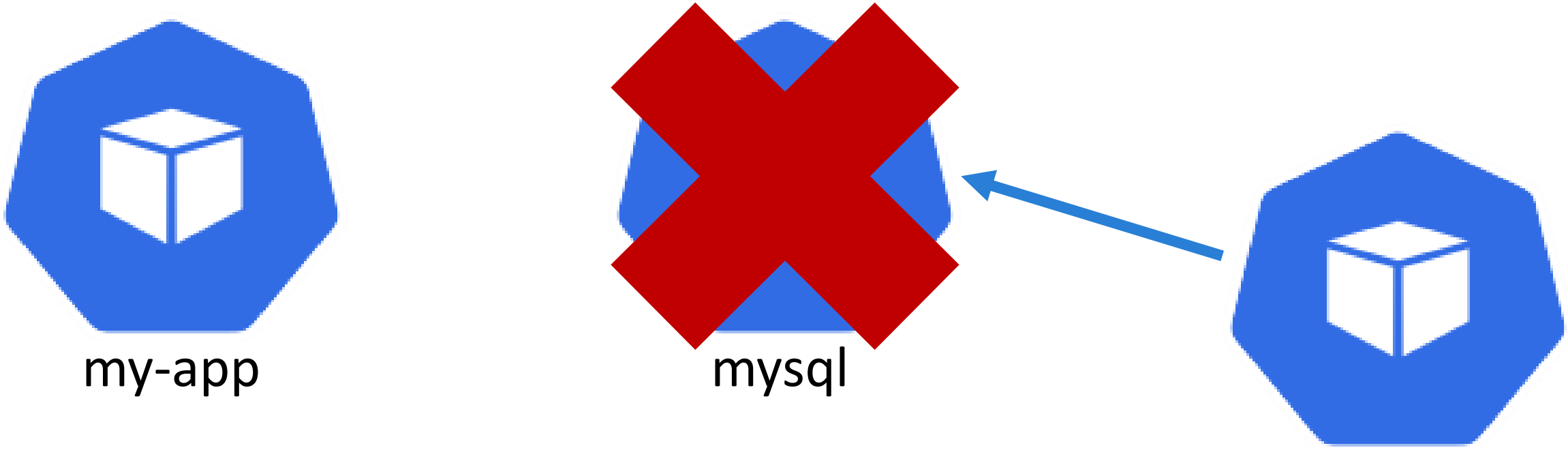


The need for Volumes



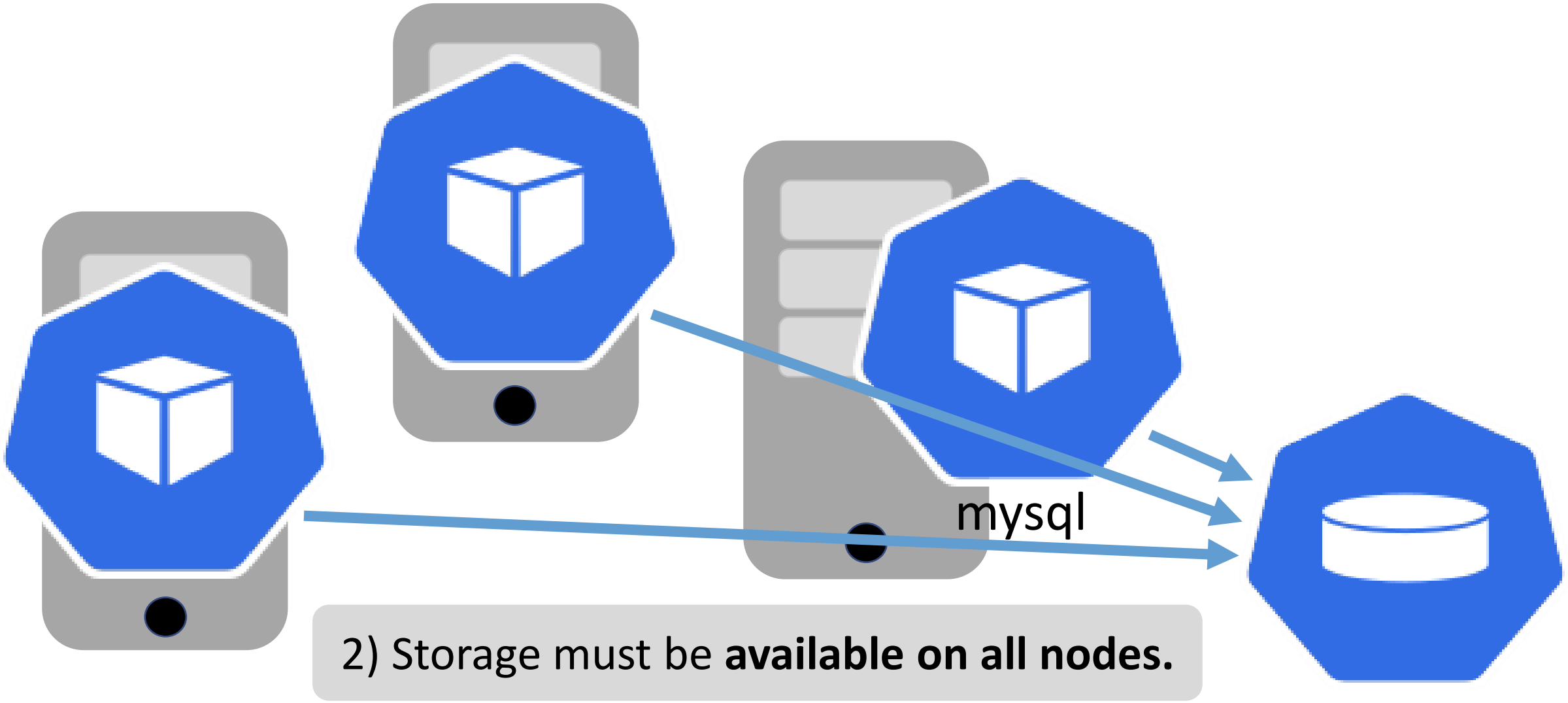
No data persistence out of the box!

Storage Requirements

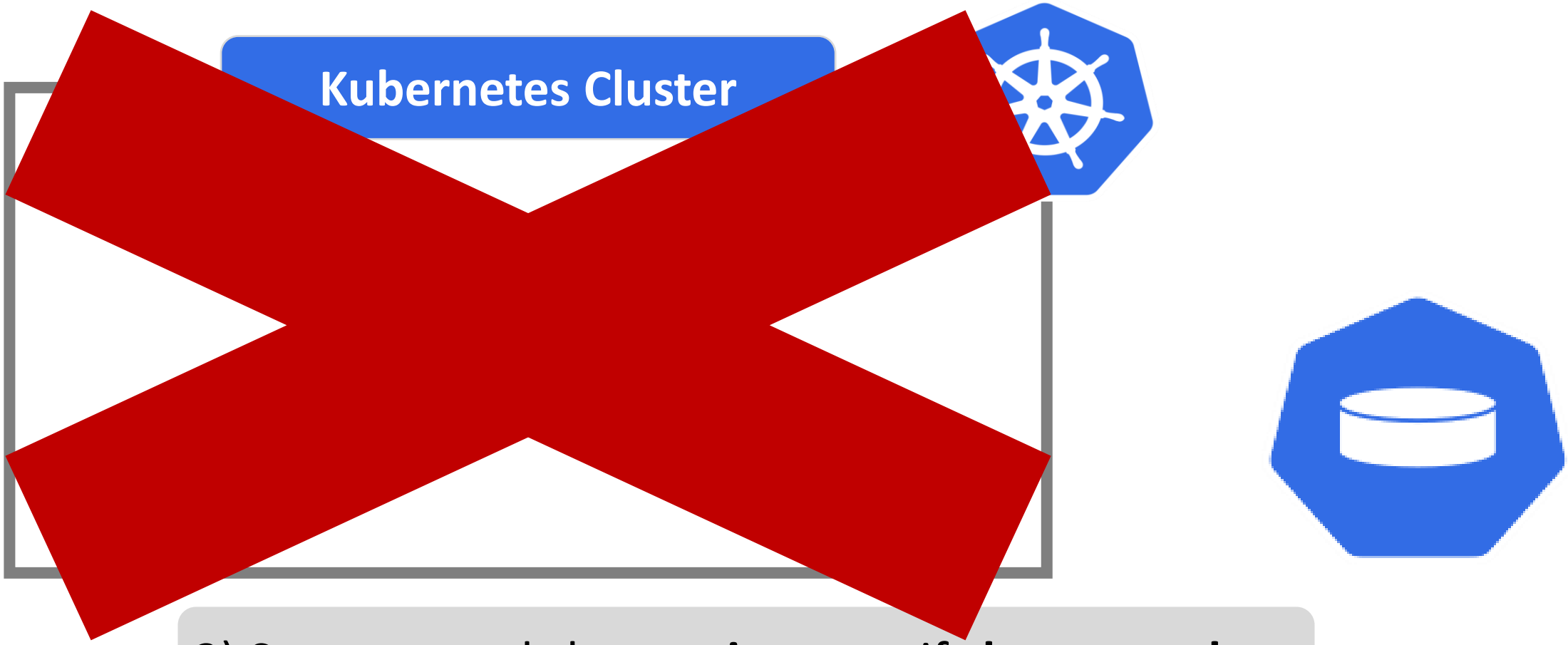


1) Storage that **doesn't depend** on the **pod lifecycle**.

Storage Requirements

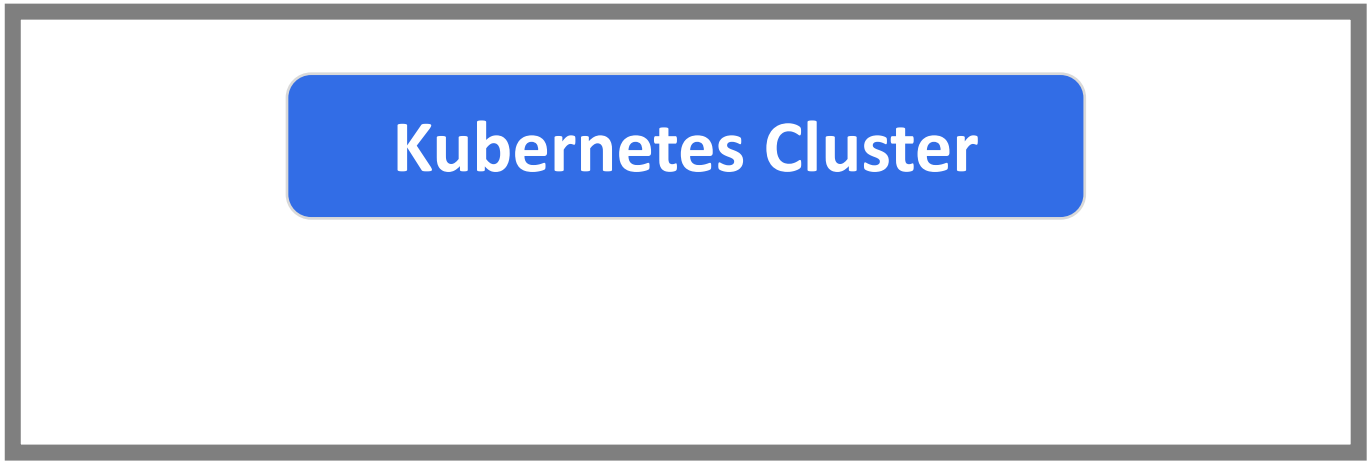


Storage Requirements



3) Storage needs be **survive** even if **cluster crashes.**

Persistent Volume



- a cluster resource

CPU

RAM

Persistent Volume

- a cluster resource

- created via YAML file

  - kind : PersistentVolume

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-name
spec:
  capacity:
    storage: 5Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Recy
```

Persistent Volume

- a cluster resource

- created via YAML file

  - kind : PersistentVolume

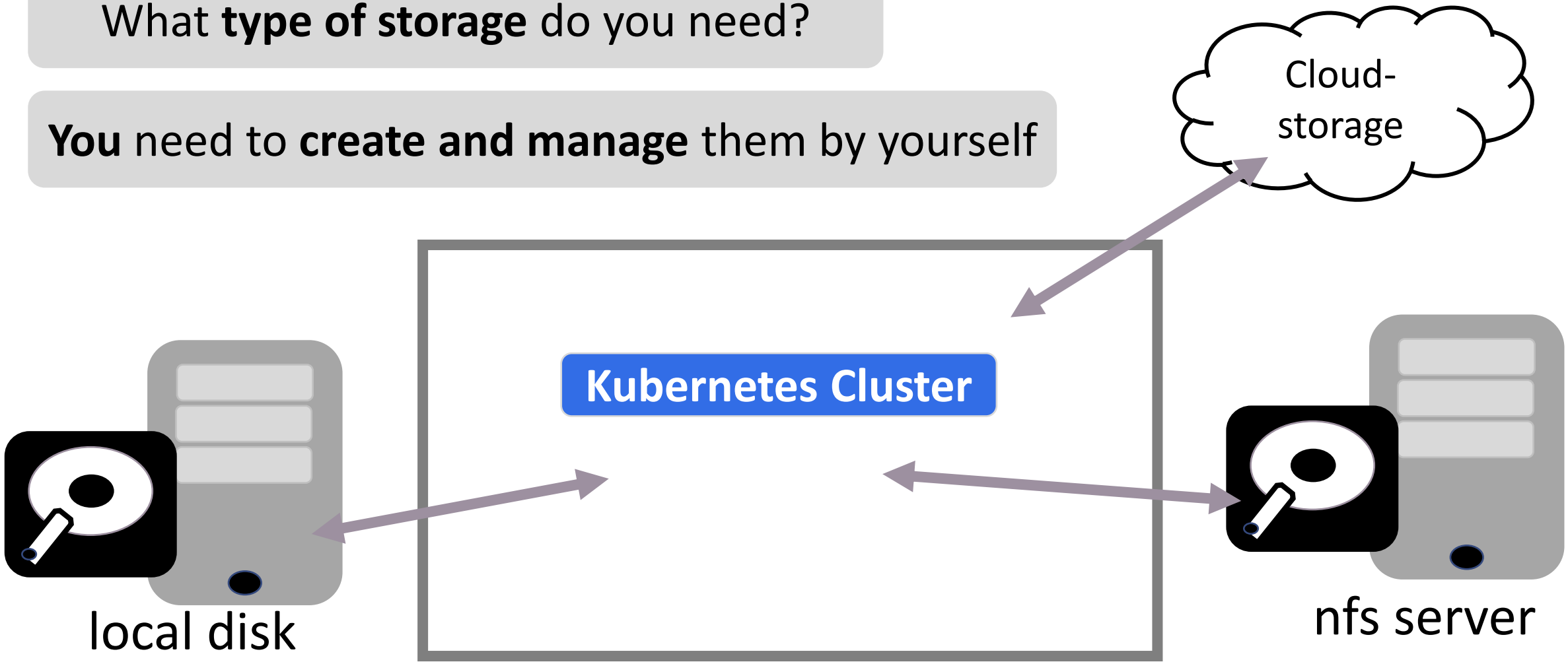
  - spec : e.g. how much storage?

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-name
spec:
  capacity:
    storage: 5Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Recy
```

Persistent Volume

What **type of storage** do you need?

You need to **create and manage** them by yourself



Persistent Volume YAML Example

Use that physical storages in the **spec** section

How much :

Additional params, like access :

Nfs parameters :

NFS Storage

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-name
spec:
  capacity:
    storage: 5Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Recycle
  storageClassName: slow
  mountOptions:
    - hard
    - nfsvers=4.0
  nfs:
    path: /dir/path/on/nfs/server
    server: nfs-server-ip-address
```

Persistent Volume YAML Example

Google Cloud

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: test-volume
  labels:
    failure-domain.beta.kubernetes.io/zone: us-central1-a__us-central1-b
spec:
  capacity:
    storage: 400Gi
  accessModes:
  - ReadWriteOnce
  gcePersistentDisk:
    pdName: my-data-disk
    fsType: ext4
```

How much :

Additional params, like access :

Persistent Volume YAML Example

Depending on storage type, spec attributes differ

Node Affinity :

Local storage

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: example-pv
spec:
  capacity:
    storage: 100Gi
  volumeMode: Filesystem
  accessModes:
  - ReadWriteOnce
  persistentVolumeReclaimPolicy: Delete
  storageClassName: local-storage
  local:
    path: /mnt/disks/ssd1
  nodeAffinity:
    required:
      nodeSelectorTerms:
      - matchExpressions:
        - key: kubernetes.io/hostname
          operator: In
          values:
          - example-node
```

Persistent Volume YAML Example

Depending on storage type, spec attributes differ

Complete list of storage backends supported by k8s :

Documentation Blog Training Partners Community Ca

the Pod must independently specify where to mo

Types of Volumes

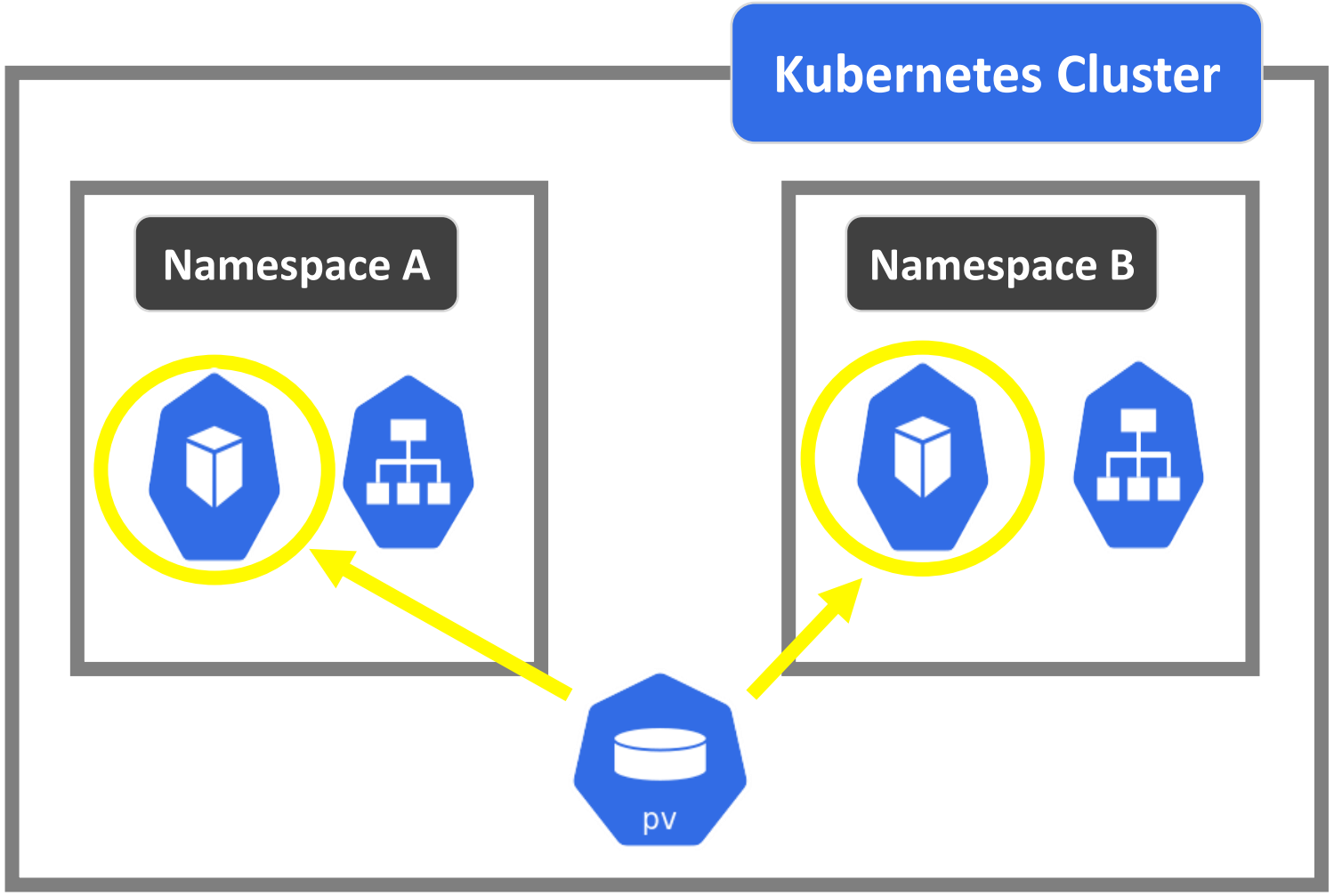
Kubernetes supports several types of Volumes:

- awsElasticBlockStore
- azureDisk
- azureFile
- cephfs
- cinder
- configMap
- csi
- downwardAPI
- emptyDir
- fc (fibre channel)
- flexVolume

Persistent Volumes are NOT namespaced

PV outside of the namespaces

Accessible to the whole cluster



Local vs. Remote Volume Types

Each volume type has it’s own use case!

Local volume types violate 2. and 3. requirement for data persistence :

- ✗ Being tied to 1 specific node
- ✗ surviving cluster nodes

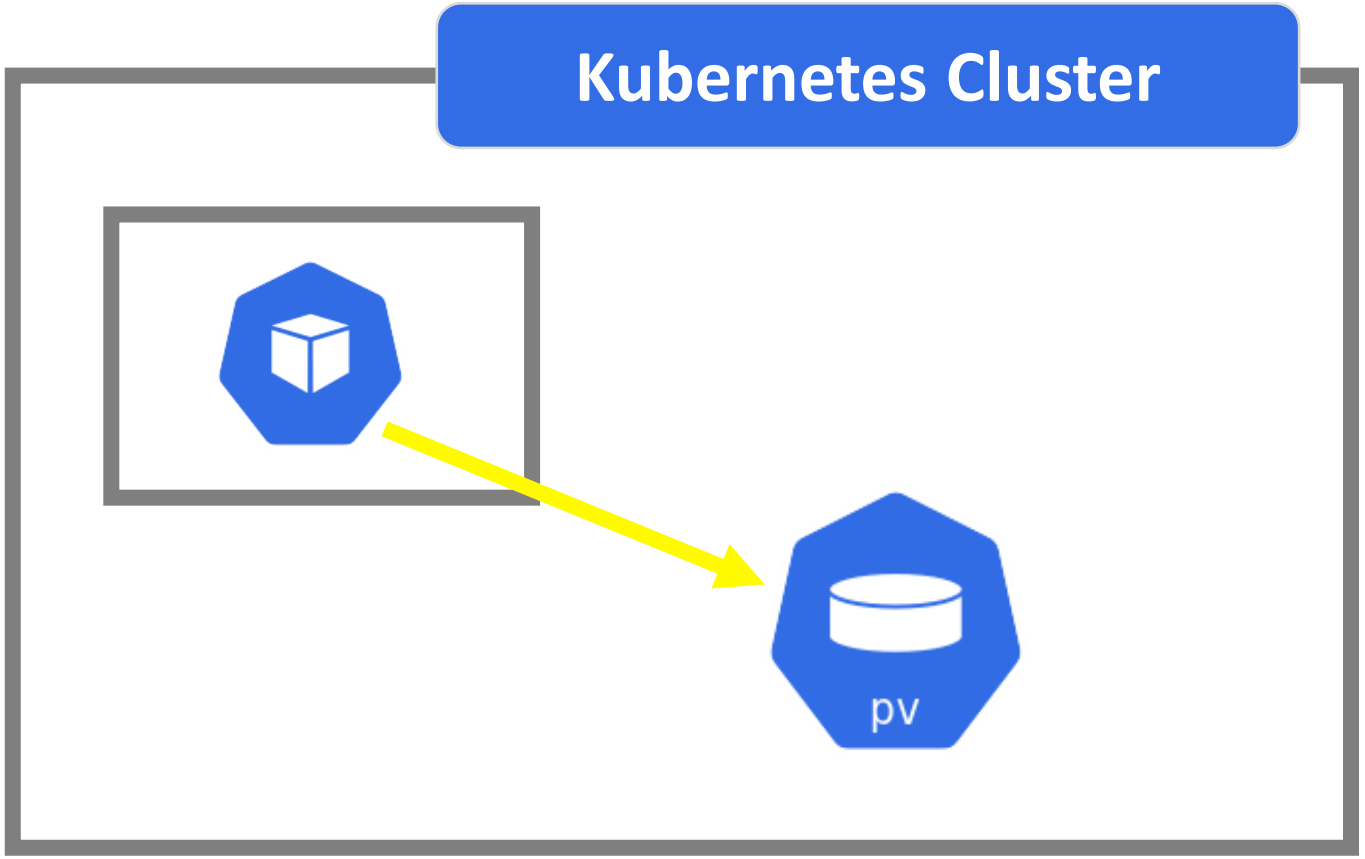
For DB persistence use remote storage!

K8s Adiministrator and K8s User

Who creates the Persistent Volumes and when?

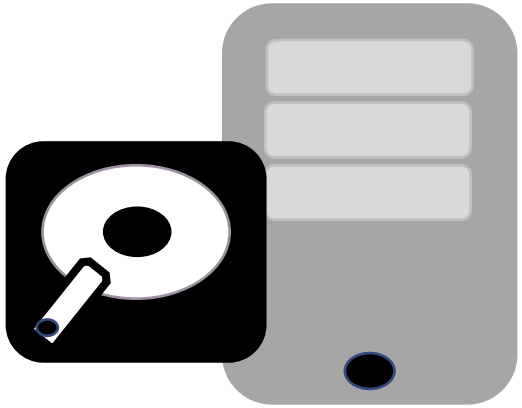
..the Pod that **depends on** it is created

PV are resources that need to be there **BEFORE..**



K8s Administrator and K8s User

Storage resource is Provisioned by Admin.



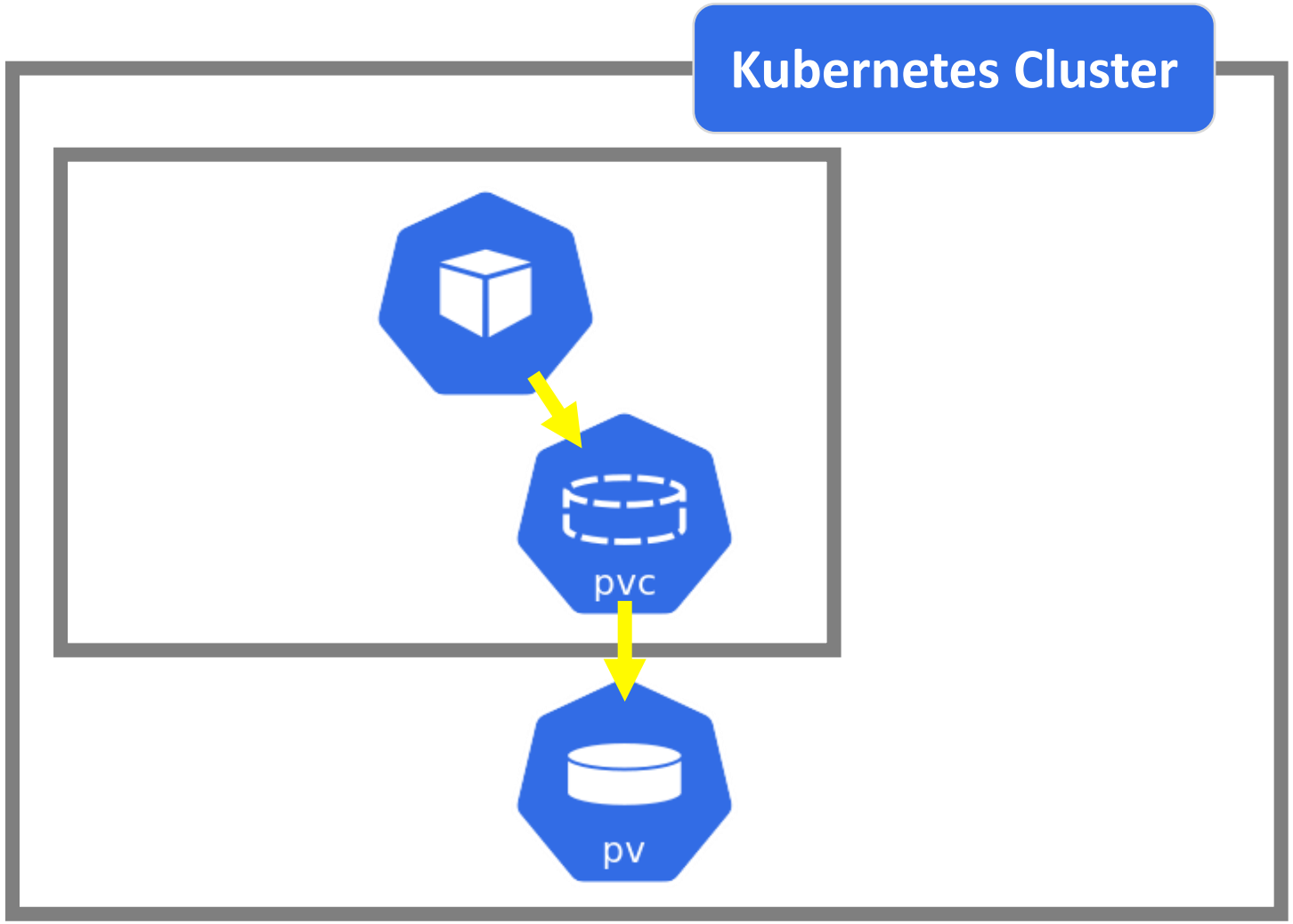
nfs-storage

Kubernetes Cluster

Cloud-storage

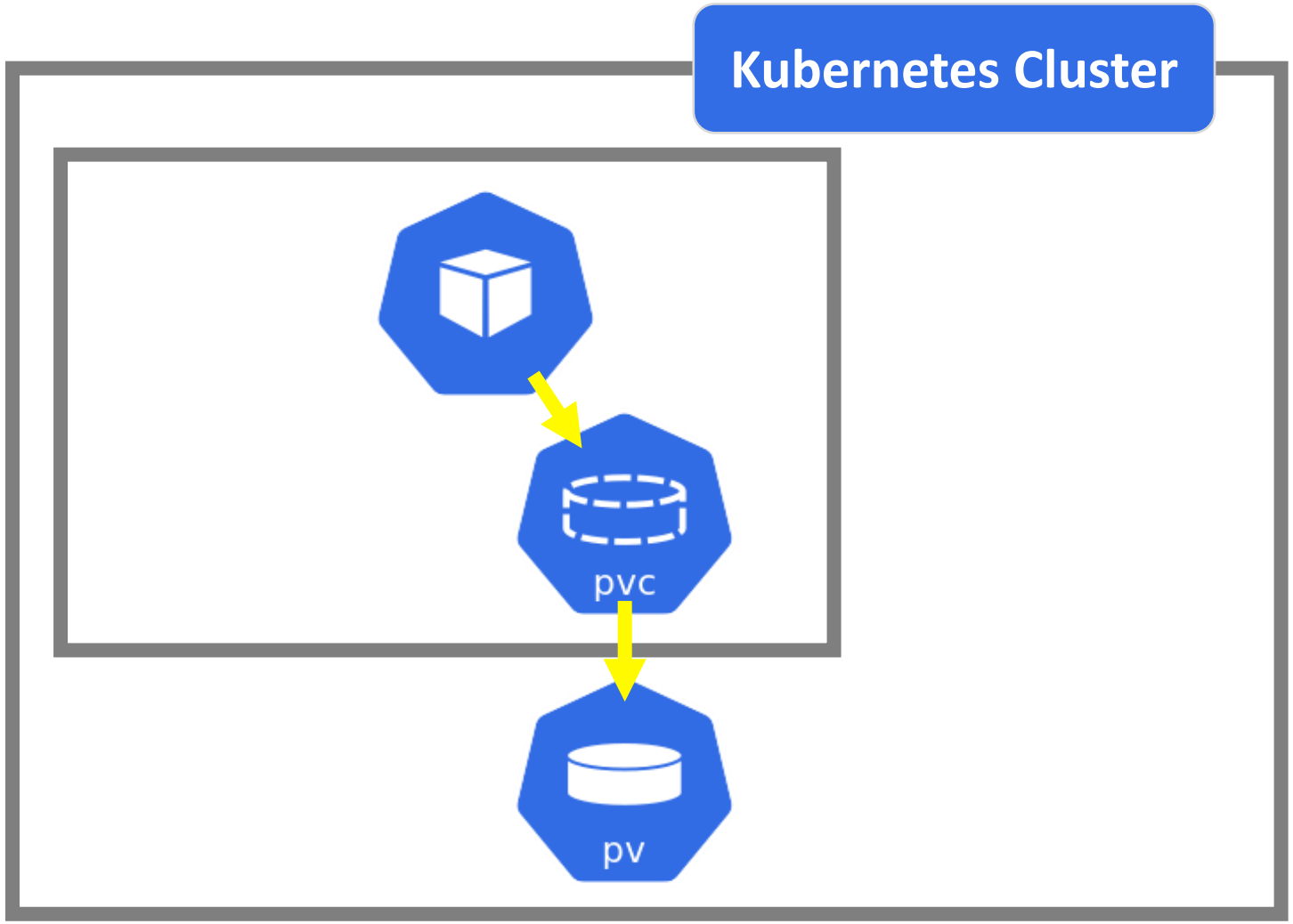
Creates the PV components from these storage backends.

Persistent Volume Claim component



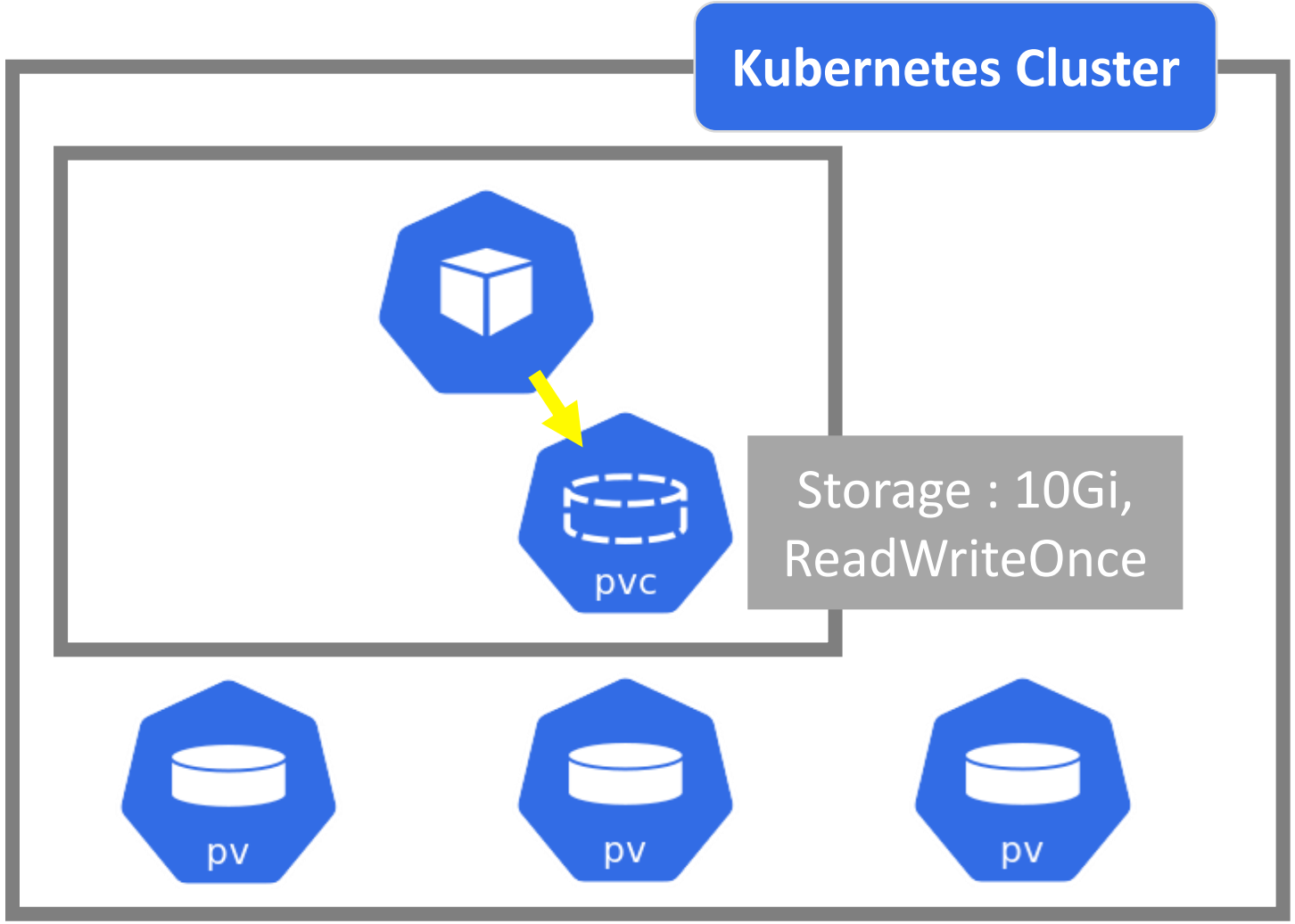
Application has to **claim** the Persistent Volume

Persistent Volume Claim component



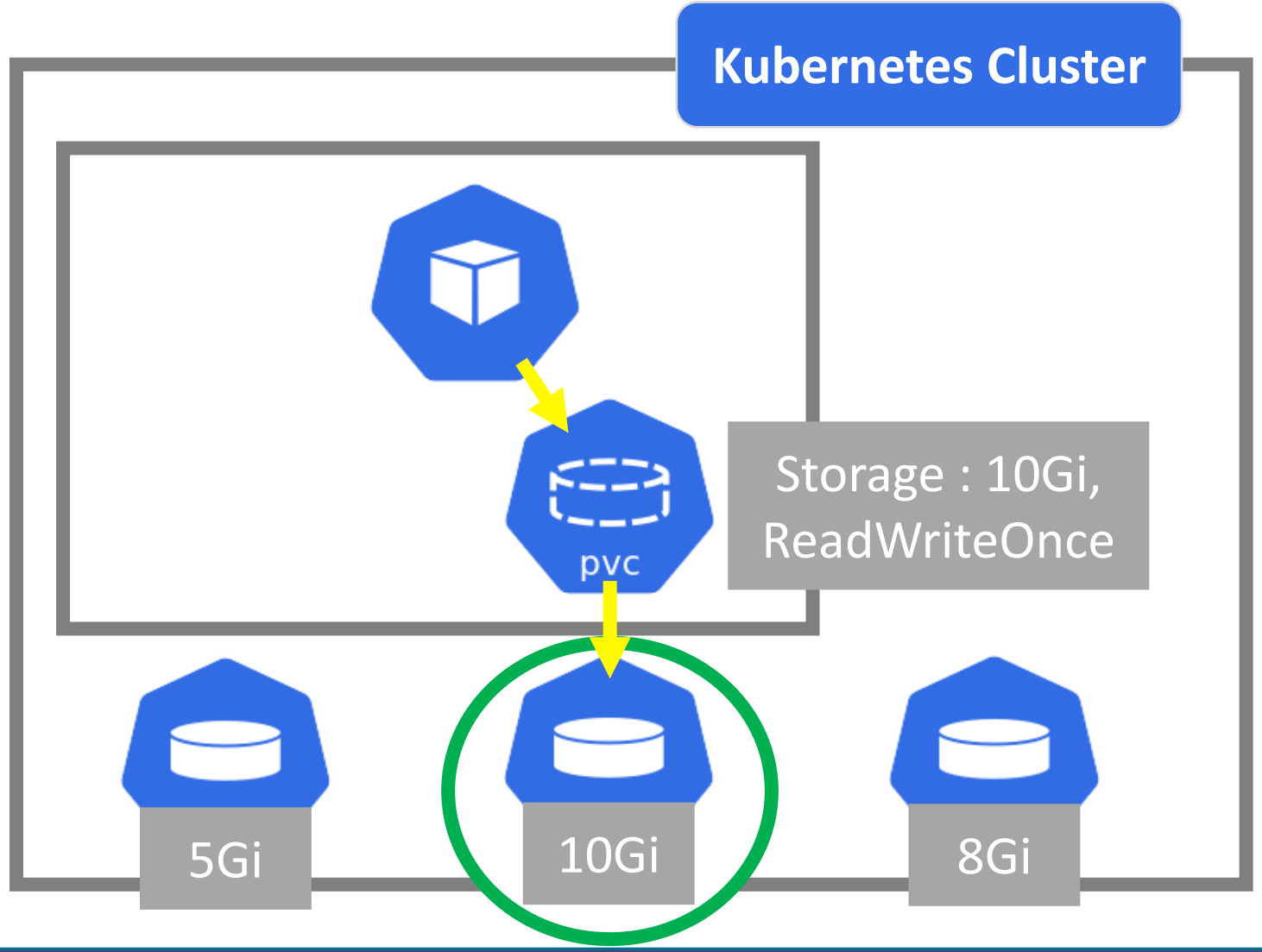

```yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc-name
spec:
  storageClassName: manual
  volumeMode: Filesystem
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
```

Persistent Volume Claim component




```yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc-name
spec:
  storageClassName: manual
  volumeMode: Filesystem
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
```

Persistent Volume Claim component




```yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc-name
spec:
  storageClassName: manual
  volumeMode: Filesystem
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
```

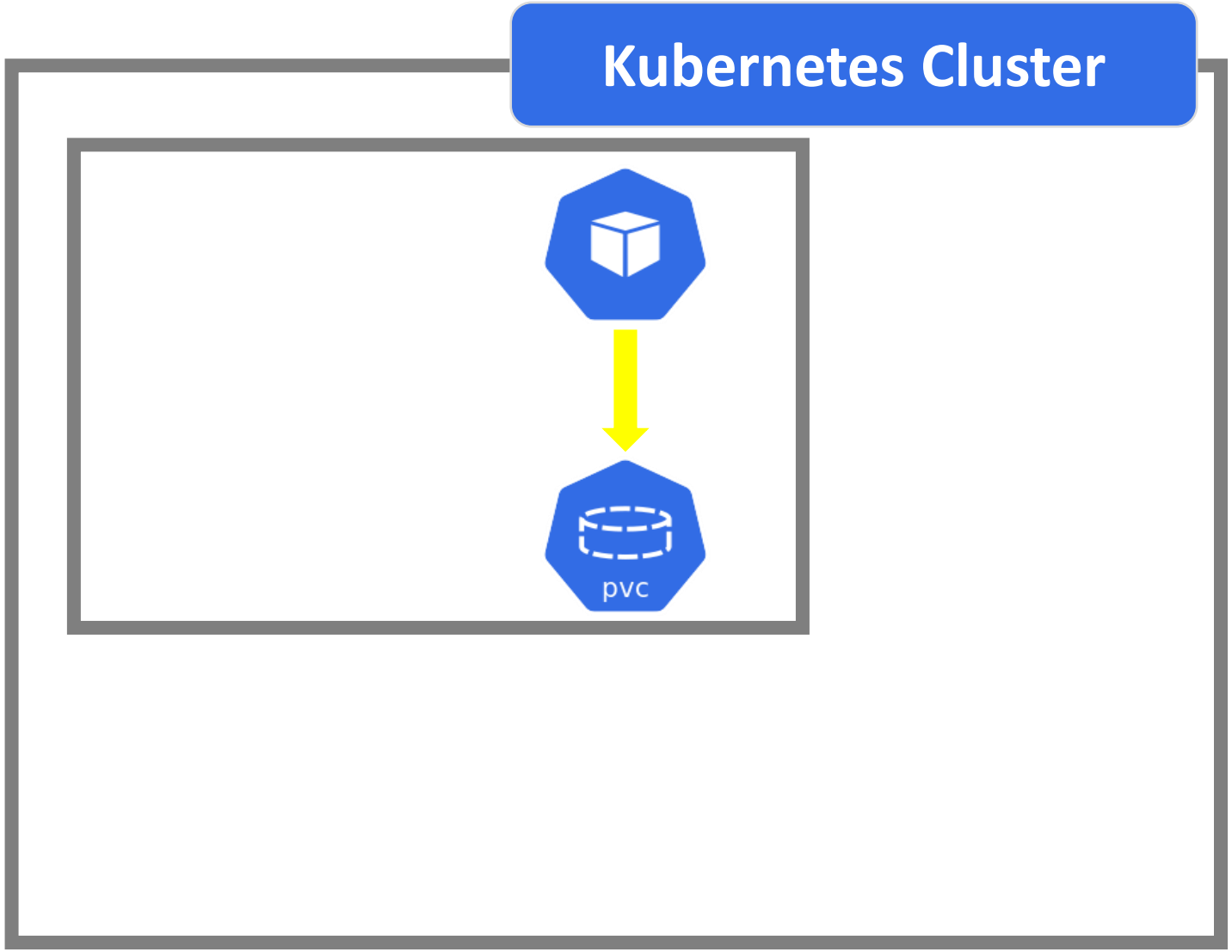
Persistent Volume Claim component

Use that PVC in Pods configuration

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: mypod
spec:
  containers:
    - name: myfrontend
      image: nginx
      volumeMounts:
      - mountPath: "/var/www/html"
        name: mypd
  volumes:
    - name: mypd
      persistentVolumeClaim:
        claimName: pvc-name
```

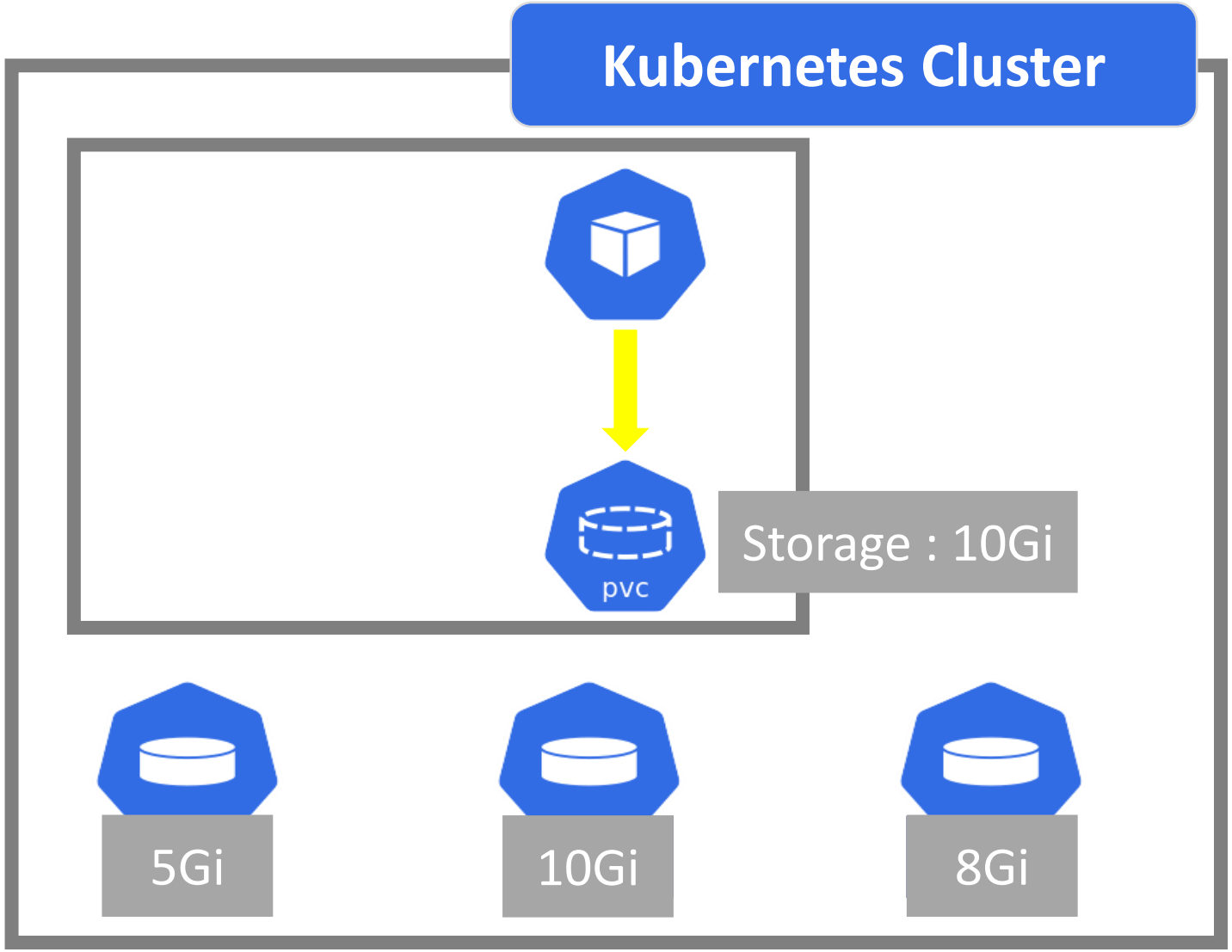

```yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc-name
spec:
  storageClassName: manual
  volumeMode: Filesystem
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
```

Levels of Volume abstraction



Pod requests the volume through the PV claim

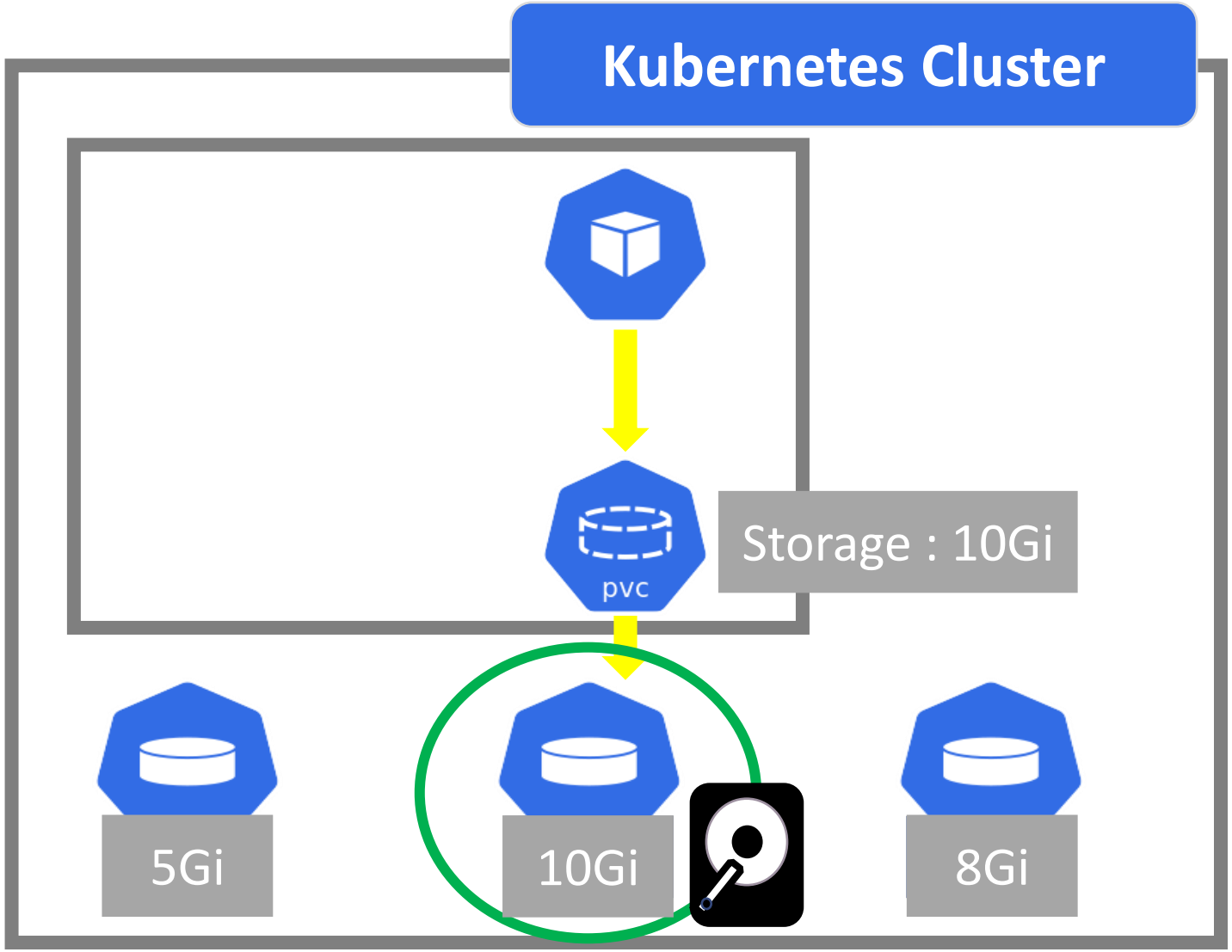
Levels of Volume abstraction



Pod requests the volume through the PV claim

Claim tries to find volume in cluster

Levels of Volume abstraction

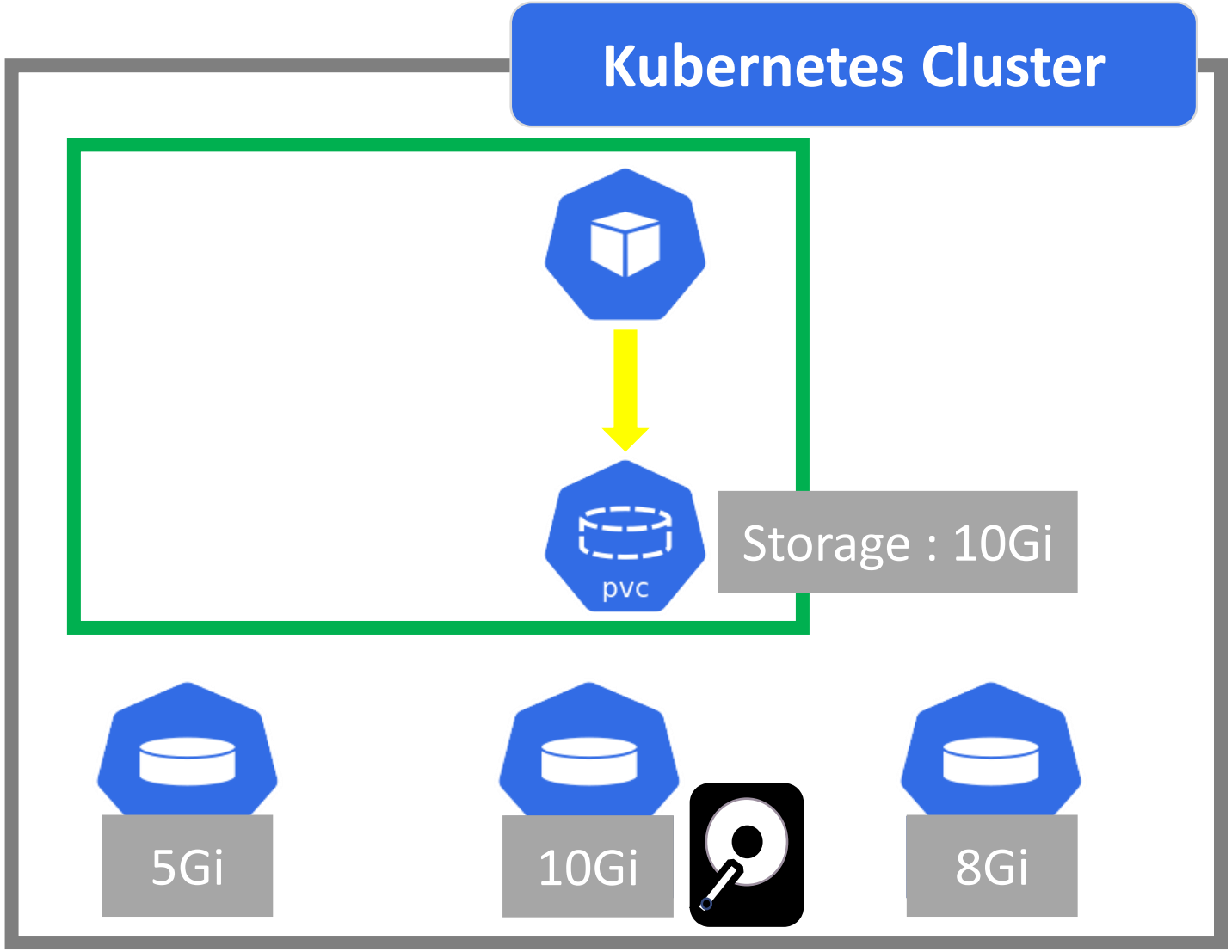


Pod requests the volume through the PV claim

Claim tries to find volume in cluster

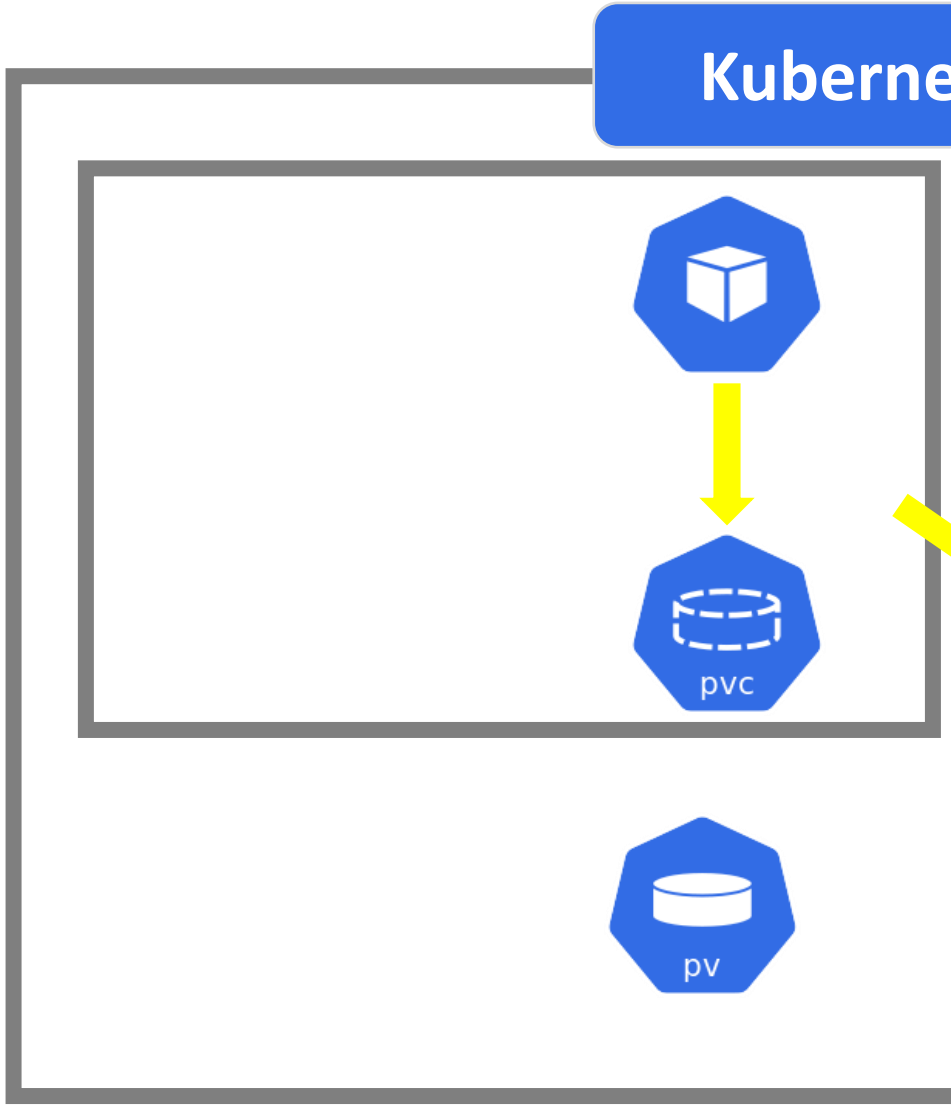
Volume has the actual storage backend

Levels of Volume abstraction



Claim must be in the **same namespace!**

Levels of Volume abstraction

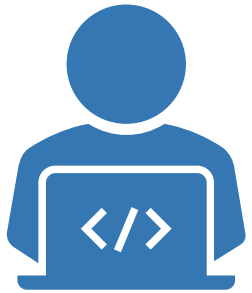



```yaml
apiVersion: v1
kind: Pod
metadata:
  name: mypod
spec:
  containers:
    - name: myfrontend
      image: nginx
      volumeMounts:
      - mountPath: "/var/www/html"
        name: mypd
  volumes:
    - name: mypd
      persistentVolumeClaim:
        claimName: pvc-name
```

Volume is **mounted** into **Container**

Volume is **mounted** into the Pod

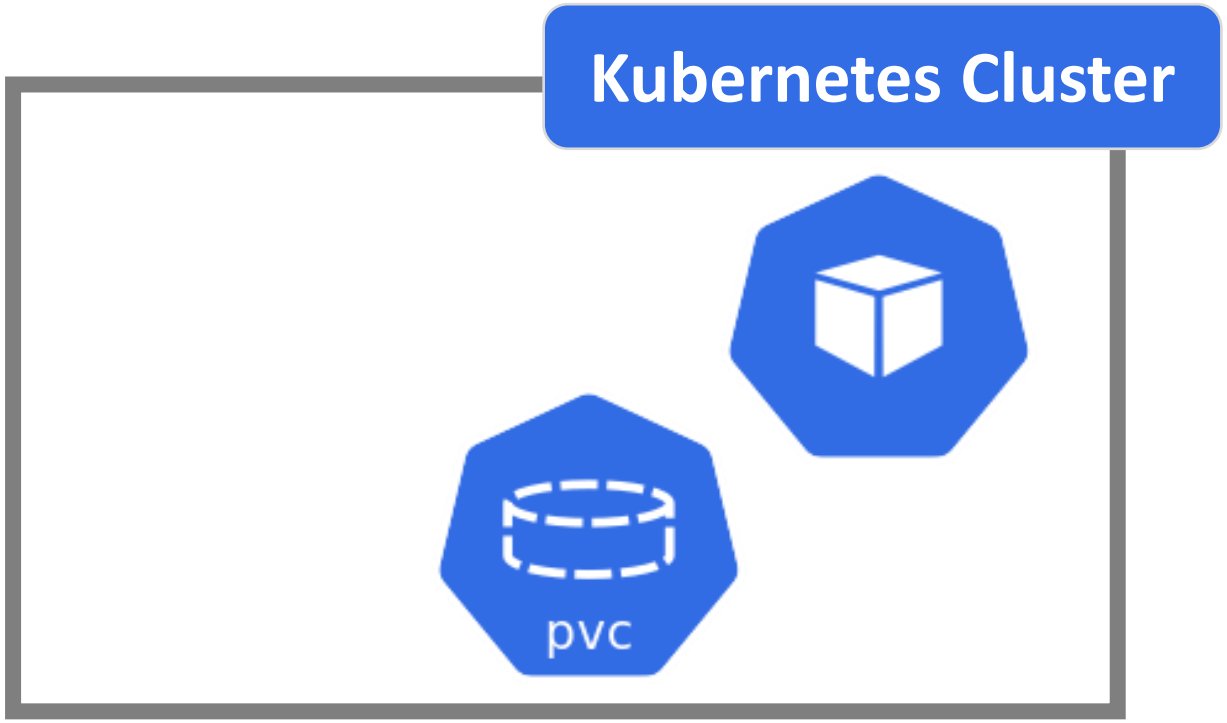
Why so many abstractions?



Admin provisions storage resource

User creates claim to PV

Why so many abstractions?



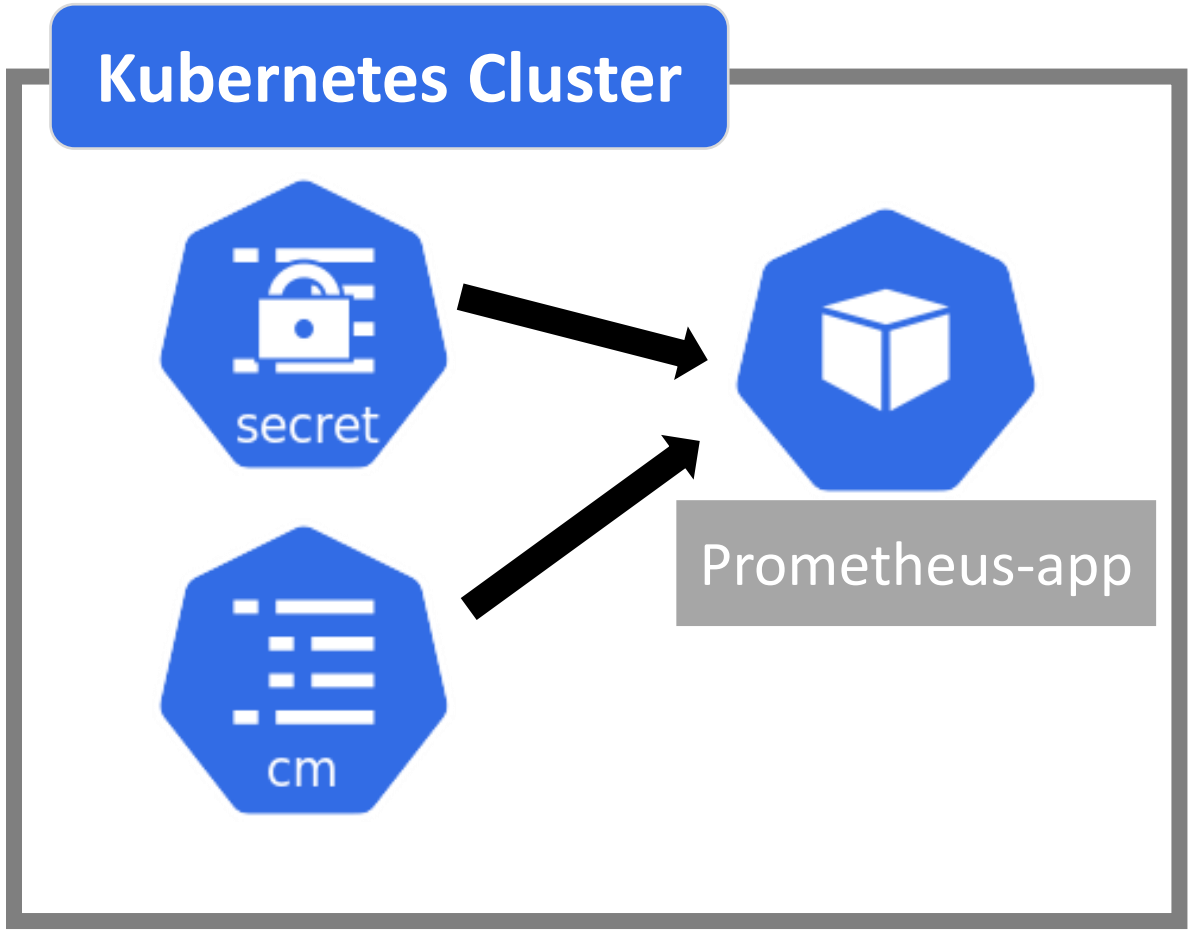
Data should be safely stored

Don’t want to set up the actual storages

ConfigMap and Secret

- Local volumes
- Not created via PV and PVC
- Managed by Kubernetes

ConfigMap and Secret



Configuration file for your pod

Certificate file for your pod

ConfigMap and Secret

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: mypod
spec:
  containers:
    - name: busybox-container
      image: busybox
      volumeMounts:
        - name: config-dir
          mountPath: /etc/config
  volumes:
    - name: config-dir
      configMap:
        name: bb-configmap
```

1) Create ConfigMap and/or Secret component

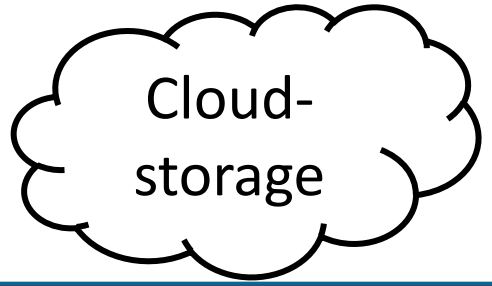
2) Mount that into your pod/container

What we've covered so far

Volume is directory with some data

These volumes are accessible in containers in a pod

How made available, backed by which storage medium
-defined by specific volume types



Types of Volumes

Kubernetes supports several types of Volumes:

- awsElasticBlockStore
- azureDisk
- azureFile
- cephfs
- cinder
- configMap
- csi
- downwardAPI
- emptyDir
- fc (fibre channel)
- flexVolume

What we've covered so far

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: mypod
spec:
  containers:
    - name: myfrontend
      image: nginx
      volumeMounts:
      - mountPath: "/var/www/html"
        name: mypd
  volumes:
    - name: mypd
      persistentVolumeClaim:
        claimName: pvc-name
```

Pod specifies what Volumes to provide

What we've covered so far

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: mypod
spec:
  containers:
    - name: myfrontend
      image: nginx
      volumeMounts:
      - mountPath: "/var/www/html"
        name: mypd
  volumes:
    - name: mypd
      persistentVolumeClaim:
        claimName: pvc-name
```

Where to mount those in the container?

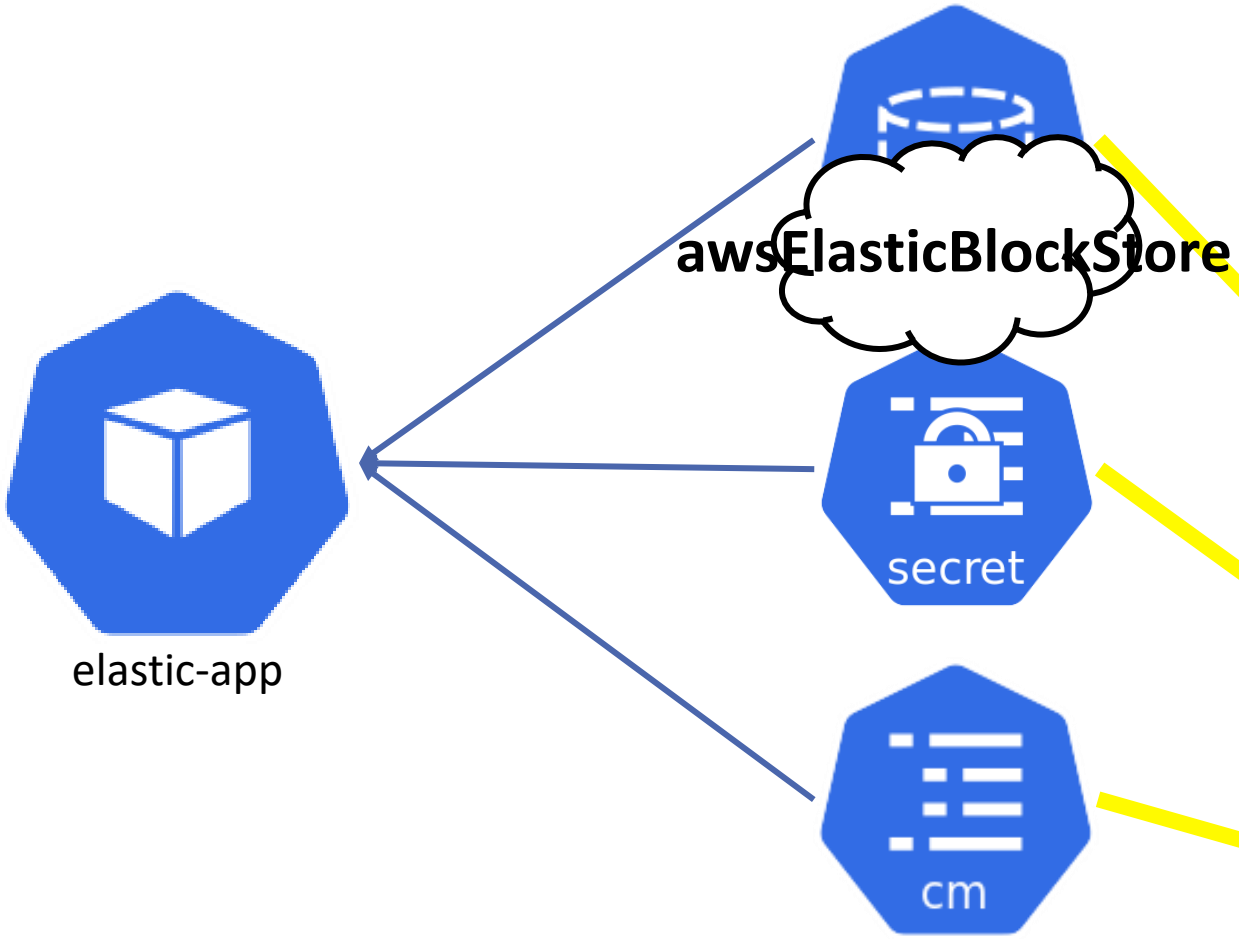
Pod specifies what Volumes to provide

What we've covered so far

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: mypod
spec:
  containers:
    - name: myfrontend
      image: nginx
      volumeMounts:
      - mountPath: "/var/www/html"
        name: mypd
  volumes:
    - name: mypd
      persistentVolumeClaim:
        claimName: pvc-name
```

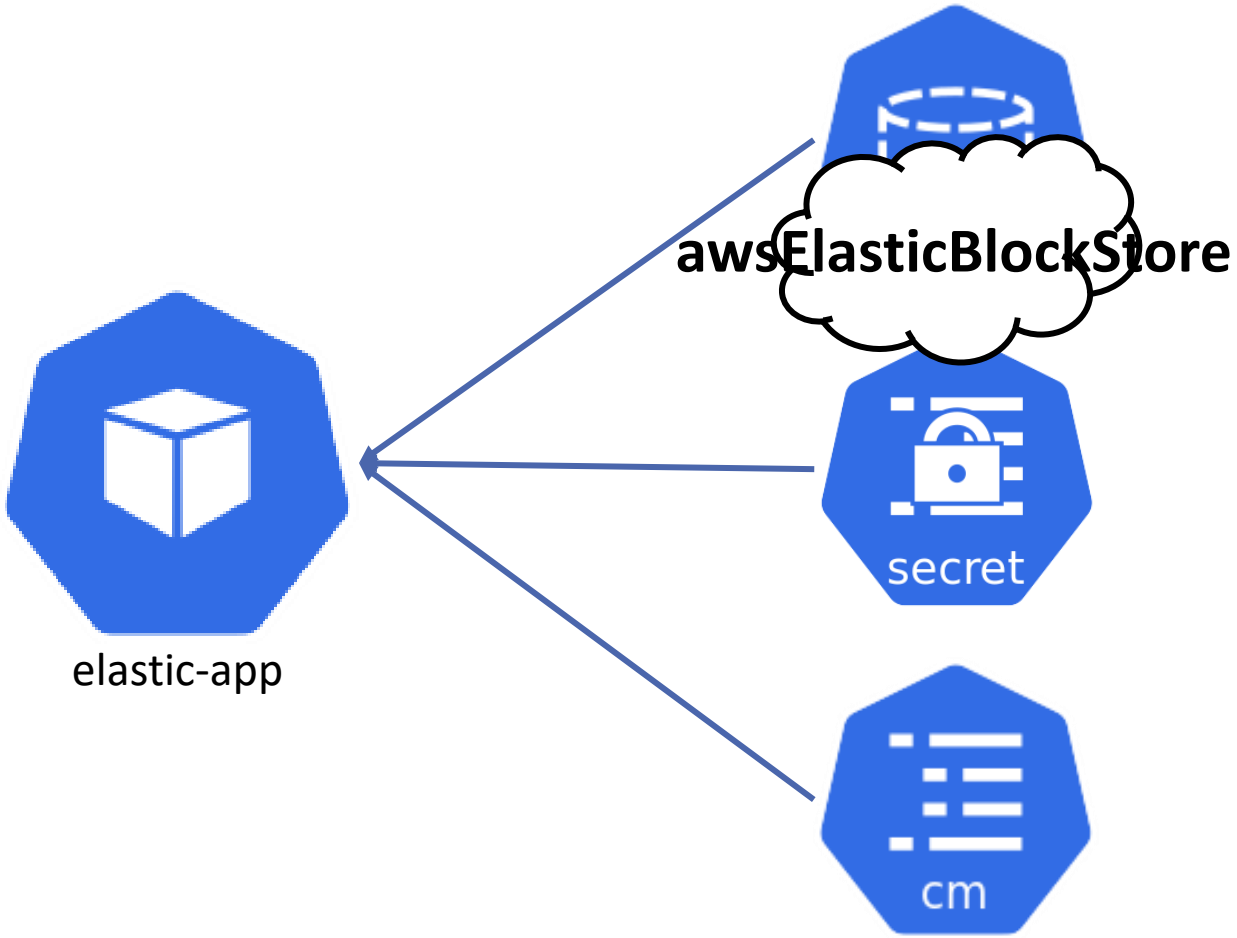
Apps can access the mounted data here: "/var/www/html"

Multiple and Different Volume Types in 1 Pod




```yaml
spec:
  containers:
  - image: elastic:latest
    name: elastic-container
    ports:
    - containerPort: 9200
    volumeMounts:
    - name: es-persistent-storage
      mountPath: /var/lib/data
    - name: es-secret-dir
      mountPath: /var/lib/secret
    - name: es-config-dir
      mountPath: /var/lib/config
volumes:
  name: es-persistent-storage
  persistentVolumeClaim:
    claimName: es-pv-claim
- name: es-secret-dir
  secret:
    secretName: es-secret
- name: es-config-dir
  configMap:
    name: es-config-map
```

Multiple and Different Volume Types in 1 Pod




```yaml
spec:
  containers:
  - image: elastic:latest
    name: elastic-container
    ports:
    - containerPort: 9200
    volumeMounts:
    - name: es-persistent-storage
      mountPath: /var/lib/data
    - name: es-secret-dir
      mountPath: /var/lib/secret
    - name: es-config-dir
      mountPath: /var/lib/config
  volumes:
  - name: es-persistent-storage
    persistentVolumeClaim:
      claimName: es-pv-claim
  - name: es-secret-dir
    secret:
      secretName: es-secret
  - name: es-config-dir
    configMap:
      name: es-config-map
```

Storage Class

1. Admins configure storage

2. Create Persistent Volumes

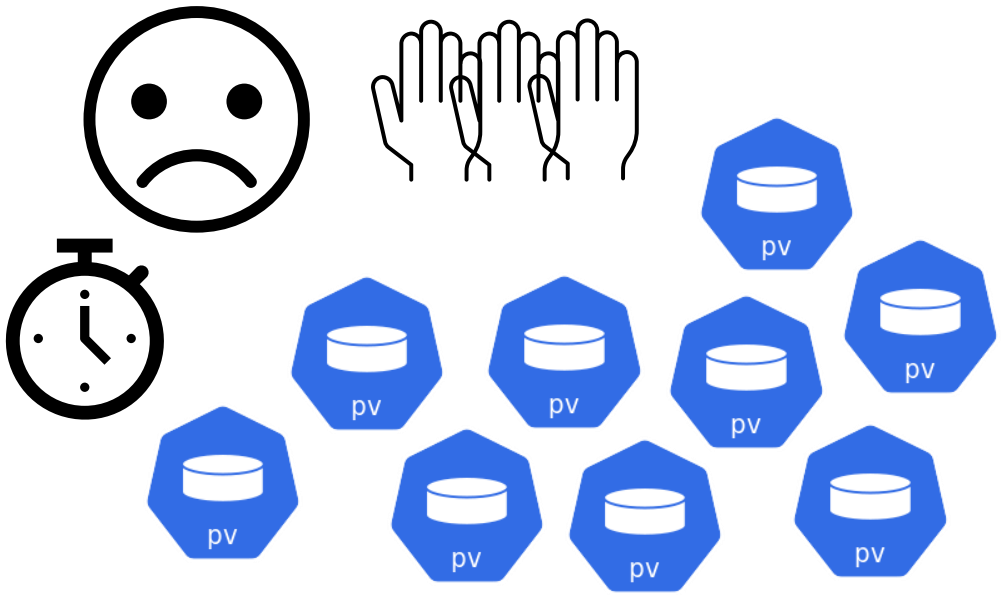
3. K8s Users claim PV using PVC

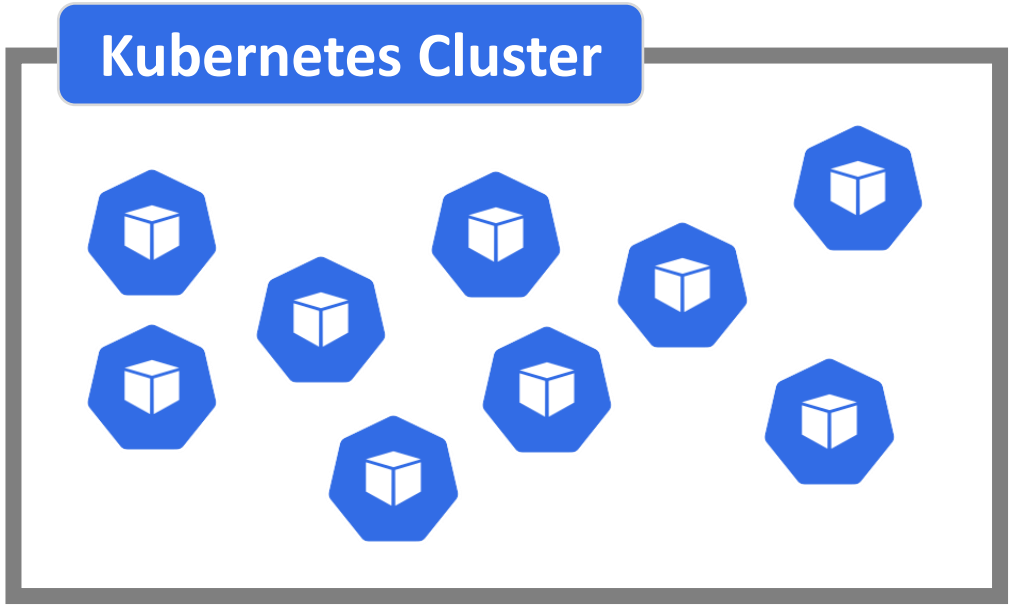
Storage Class

Consider a cluster with hundreds of applications where things get deployed and storage is needed for these applications

- Admins may have to manually request storage from cloud / storage provider and create hundreds of PV for all the apps

That can be tedious time-consuming and can get messy very quickly

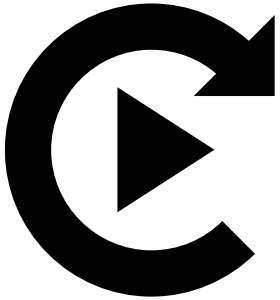




Storage Class

SC provisions Persistent Volumes **dynamically..**

..when PersistentVolumeClaim claims it



Storage Class

kind : StorageClass

```yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: storage-class-name
provisioner: kubernetes.io/aws-ebs
parameters:
  type: io1
  iopsPerGB: "10"
  fsType: ext4
```

Storage Class

StorageBackend is defined in the SC component

- via “**provisioner**” attribute

- each storage backend has own provisioner

- **Internal** provisioner – “Kubernetes.io”

- **external** provisioner

- configure **parameters** for storage we want to request for PV

```yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: storage-class-name
provisioner: kubernetes.io/aws-ebs
parameters:
  type: io1
  iopsPerGB: "10"
  fsType: ext4
```

Storage Class

Another abstraction level

- abstracts underlying storage provider

- parameters for that storage

```yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: storage-class-name
provisioner: kubernetes.io/aws-ebs
parameters:
  type: io1
  iopsPerGB: "10"
  fsType: ext4
```

Storage Class

Requested by PersistentVolumeClaim

Storage Class Config

```yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: storage-class-name
provisioner: kubernetes.io/aws-ebs
parameters:
  type: io1
  iopsPerGB: "10"
  fsType: ext4
```

PVC Config

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
    name: mypvc
spec:
    accessModes:
    - ReadWriteOnce
    resources:
      requests:
        storage: 100Gi
    storageClassName: storage-class-name
```

Persistent Volume

Persisent Volume Claim

Storage Class