Working with Volumes in Kubernetes

Patrick Rusch
Senior Consultant / IT Instructor

Volumes in Kubernetes: Types and Use Cases

Why Do We Need Volumes?

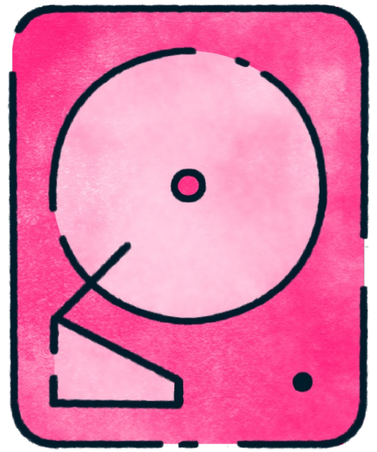
Containers are ephemeral

Data is lost on restarts

Volumes allow persistence and data sharing

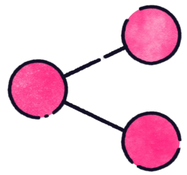
Ephemeral vs. Persistent Volumes

Ephemeral volumes, like emptyDir, configMap, secret



Persistent volumes

Common Ephemeral Volumes



emptyDir: shared scratch space

configMap: inject config

secret: inject sensitive data

Persistent Volumes and Claims

Persistent Volume is a resource in the cluster

Persistent Volume Claims is a request for that storage

Essential for not losing any data

For temporary files: emptyDir works well

For configuration: use configMap or secret

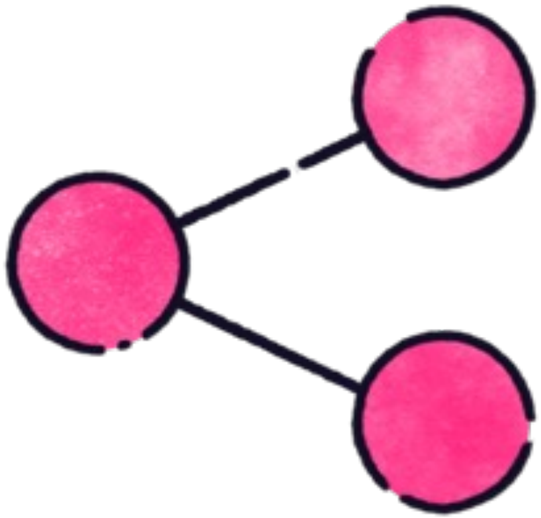
For durable data: go with Persistent Volumes and Claims

emptyDir, configMap, and secret: Ephemeral Storage

What Are Ephemeral Volumes?

- Tied to the pod’s lifecycle
- Data is deleted when the pod is deleted
- Used for temporary storage or data sharing

emptyDir: Temporary Storage



Created when the pod starts

Deleted when the pod is removed

Useful for scratch space

Use emptyDir in Pod Spec

example-pod.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: example-pod
spec:
  containers:
  - name: app1
    image: app1-image
    volumeMounts:
    - name: temp-storage
      mountPath: /data
  volumes:
  - name: temp-storage
    emptyDir: {}
```

configMap: Injecting Configuration

Stores non-sensitive configuration

Easily injects config into pods

Avoids rebuilding the container image

Example of configMap

app-config.yaml

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: app-config
data:
  APP_MODE: production
  LOG_LEVEL: debug
```

secret: Injecting Sensitive Data

Stores sensitive information (e.g., passwords, tokens)

Encoded in base64 for basic security

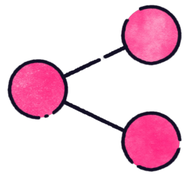
Can be mounted as a volume or environment variable

Example of Secret

db-password.yaml

```
apiVersion: v1
kind: Secret
metadata:
  name: db-password
data:
  password: cGFzc3dvcmQ=  # "password" encoded in base64
```

When to Use Ephemeral Volumes



Temporary storage, data sharing

Inject configuration

Inject sensitive data securely

Persistent Volumes and Persistent Volume Claims

What Is a Persistent Volume (PV)?

A piece of storage provisioned by an administrator

Independent of the pod lifecycle

Can be backed by various storage types (e.g., NFS, cloud storage)

What Is a Persistent Volume Claim (PVC)?

A request for storage by a user or application

Defines storage size, access modes, and storage class

PVCs are automatically bound to PVs by Kubernetes

How PVs and PVCs Work Together

1. Admin creates PVs
2. PVs provide the physical storage
3. PVCs request specific storage
4. Kubernetes binds PVCs to matching PVs

Persistent Volume Definition

example-pv.yaml

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: example-pv
spec:
  capacity:
    storage: 5Gi
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  storageClassName: standard
  awsElasticBlockStore:
    volumeID: vol-xxxxxxxxxxxx
    fsType: ext4
```

Persistent Volume Claim Definition

example-pvc.yaml

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: example-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 5Gi
```

Using PVC in a Pod

example-pod.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: example-pod
spec:
  containers:
  - name: app
    image: app-image
    volumeMounts:
    - name: storage
      mountPath: /data
  volumes:
  - name: storage
    persistentVolumeClaim:
      claimName: example-pvc
```

StorageClasses and Dynamic Provisioning

What Is a StorageClass?

Defines characteristics of storage

Specifies how storage should be provisioned (e.g., fast SSD, standard disk)

Includes provisioning parameters like volume type, performance, etc.

Dynamic Provisioning: The Power of Automation

- Automatically provisions PVs based on PVC requests
- Saves time and reduces errors by automating storage management
- Ensures consistent, on-demand storage for applications

How Dynamic Provisioning Works

StorageClass specifies the provisioning method (e.g., EBS, GCE persistent disks)

Kubernetes automatically provisions PVs and binds them to PVCs

StorageClass Definition

standard.yaml

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: standard
provisioner: kubernetes.io/aws-ebs
parameters:
  type: gp2
  fsType: ext4
reclaimPolicy: Retain
```

Persistent Volume Claim with StorageClass

my-pvc.yaml

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: my-pvc
spec:
  storageClassName: standard
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
```

Demo: Deploy a Pod Using Both Ephemeral and Persistent Storage

Course Recap

Course Summary

Container images

Kubernetes workloads

Multi-container pods

Volumes and storage

Container Images

- Dockerfile best practices
- Updating and tagging images
- Using images in pods

Kubernetes Workloads

- Deployments vs. StatefulSets
- DaemonSets, CronJobs, and Jobs
- Real-world controller selection

Multi-container Pods

Sidecar, Init, Adapter, Ambassador

Benefits of container separation

Real-world patterns

Volumes and Storage

- Ephemeral vs. persistent storage
- ConfigMaps and secrets
- PVCs and StorageClasses

- Built smart images
- Choose the right controllers
- Used design patterns for flexibility
- Managed storage with confidence

Take It Further

- Apply in your real-world projects
- Explore CI/CD, observability, and scaling
- Keep building — Kubernetes is a journey

Thank you for your time!