CHAPTER 7

Storage in Kubernetes

As applications evolve, the demands for data storage for persistent and non-persistent data grow increasingly complex. In the dynamic environment of Kubernetes, managing persistent storage presents unique challenges, such as ensuring data consistency, handling the lifecycle of storage resources, and optimizing performance. Kubernetes offers robust solutions to these problems through its storage management features, including persistent volumes (PVs), Persistent Volume Claims (PVCs), and storage classes.

This chapter guides us into the intricacies of managing storage in Kubernetes, providing practical steps and insights for effectively leveraging these storage mechanisms. Whether we are dealing with stateful applications that require data persistence or seeking to automate storage provisioning for scalable applications, this chapter offers the guidance we need.

7.1 Storage for Containerized Workloads

Effectively managing storage for containerized workloads is crucial for ensuring data persistence, consistency, and optimal performance in dynamic and scalable environments like Kubernetes.

G. Stencel and L. Berton, *Kubernetes Recipes*,
https://doi.org/10.1007/979-8-8688-1325-2_7

Problem

Containerized applications often require persistent storage to retain data beyond the lifespan of individual containers. This persistent storage need presents challenges in dynamic and scalable environments like Kubernetes. Specifically, managing the lifecycle of storage resources, ensuring data consistency, and optimizing performance are common problems.

Solution

Kubernetes manages storage via persistent volumes (PVs), Persistent Volume Claims (PVCs), and storage classes. Here's how we can manage storage for containerized workloads:

1. **Define a Persistent Volume (PV)**: Create a YAML file to define a persistent volume, for example:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-example
spec:
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  storageClassName: standard
  hostPath:
    path: "/mnt/data"
```

2. **Define a Persistent Volume Claim (PVC)**: Create a YAML file to define a Persistent Volume Claim, for example:

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

3. **Deploy an Application Using the PVC**: Modify our application deployment YAML file to use the PVC, for example:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
```

```
  spec:
    containers:
    - name: nginx
      image: nginx:latest
      volumeMounts:
      - mountPath: "/usr/share/nginx/html"
        name: storage
    volumes:
    - name: storage
      persistentVolumeClaim:
        claimName: pvc-example
```

4. **Apply the Configurations**: Apply the PV, PVC, and application deployment configurations using `kubectl`:

```
kubectl apply -f pv.yaml
kubectl apply -f pvc.yaml
kubectl apply -f deployment.yaml
```

Discussion

Managing storage in Kubernetes involves understanding and correctly implementing persistent volumes, Persistent Volume Claims, and storage classes. PVs are storage resources in the cluster, while PVCs are requests for those resources. Storage classes provide a way to define different types of storage (e.g., fast SSDs, slow HDDs) and their provisioning parameters.

Persistent Volumes (PV)

PVs are resources in the cluster that provide durable storage. They have a lifecycle independent of any individual pod that uses the PV. They are defined by a YAML file and include details such as storage capacity and access modes (e.g., `ReadWriteOnce`, `ReadOnlyMany`, `ReadWriteMany`).

Persistent Volume Claims (PVC)

PVCs are requests for storage by a user. They are similar to a pod in that they consume resources in the cluster. PVCs can request specific sizes and access modes, and Kubernetes will bind the PVC to an available PV that meets the requirements.

Storage Classes

Storage classes allow administrators to define different types of storage offered in a cluster. Each storage class might map to a different quality-of-service level, such as IOPS performance or backup policies.

7.2 Defining StorageClass

Problem

Manually provisioning persistent volumes (PVs) can be time-consuming and error-prone, especially in dynamic environments where storage needs frequently change. This approach lacks flexibility and can lead to inefficiencies in resource utilization.

Solution

Use a **StorageClass** to enable dynamic provisioning of PVs. A **StorageClass** defines the parameters and provisioner for storage backends, allowing Kubernetes to automatically create PVs on demand based on PVC requests.

Here's an example **StorageClass** definition:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: fast-storage
```