TEL4VN

Managing Data in the Kubernetes Cluster

THAO LUONG

03/2022

Content

- ❑ Configuring Pods with Environment Variables
- ❑ Managing Application Configuration with ConfigMaps
- ❑ Working with Sensitive Data Using Secrets
- ❑ Persistent Storage and Container
- ❑ Kubernetes Storage
- ❑ Storage lifecycle
- ❑ Using storage in Kubernetes

Pod environment variable

- Pod allow us to pass some environment variable when start the pod

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: envar-demo
  labels:
    purpose: demonstrate-envars
spec:
  containers:
  - name: envar-demo-container
    image: gcr.io/google-samples/node-hello:1.0
    env:
    - name: DEMO_GREETING
      value: "Hello from the environment"
    - name: DEMO_FAREWELL
      value: "Such a sweet sorrow"
```

Configmap

- Used to store non-confidential data in key-value pairs
- Define application or environment specific setting
- Decouple application and pod config.
- Maximizing our container image portability

Use configmap

- Environment variable
- Volume and files
  - Volume mount inside a container
  - Volume config map can be updated

Define Configmap

```
kubectl create configmap appconfigprod \
 --from-literal=DATABASE_SERVERNAME=sql.example.local \
 --from-literal=BACKEND_SERVERNAME=be.example.local

kubectl create configmap appconfigqa \
 --from-file=appconfigqa

apiVersion: v1
kind: ConfigMap
metadata:
 name: appconfigprod
data:
  BACKEND_SERVERNAME: be.example.local
  DATABASE_SERVERNAME: sql.example.local
```

Using Configmap in Environment Variable

```yaml
containers:
- name: hello-world
  ...
  env:
  - name: DATABASE_SERVERNAME
    valueFrom:
      configMapKeyRef:
        name: appconfigprod
        key: DATABASE_SERVERNAME
  - name: BACKEND_SERVERNAME
    valueFrom:
      configMapKeyRef:
        name: appconfigprod
        key: BACKEND_SERVERNAME
```

```yaml
containers:
- name: hello-world
  ...
  envFrom:
    - configMapRef:
        name: appconfigprod
```

Using Configmap as File

```
spec:
  volumes:
    - name: appconfig
      configMap:
        name: appconfigqa
  containers:
  - name: hello-world
    ...
    volumeMounts:
      - name: appconfig
        mountPath: "/etc/appconfig"
```

Secrets

- object that contains a small amount of sensitive data such as a password, a token, or a key
- More flexible and secure when store confident information

Usage Secrets

- Environment variable
- Volumes of files
- By kubelet when pulling images for pod

Creating Secrets

```
kubectl create secret generic app1 \
 --from-literal=USERNAME=app1login \
 --from-literal=PASSWORD='S0methingS@Str0ng!'
```

Using Secrets in Environment Variable

```yaml
spec:
  containers:
  - name: hello-world
    ...
    env:
    - name: app1username
      valueFrom:
        secretKeyRef:
          name: app1
          key: USERNAME
    - name: app1password
      valueFrom:
        secretKeyRef:
          name: app1
          key: PASSWORD
```

```yaml
spec:
  containers:
  - name: hello-world
    ...
    envFrom:
    - secretRef:
        name: app1
```

Using Secrets as File

```
spec:
  volumes:
    - name: appconfig
      secret:
        secretName: app1
  containers:
    ...
    volumeMounts:
      - name: appconfig
        mountPath: "/etc/appconfig"
```

```
/etc/appconfig/USERNAME
/etc/appconfig/PASSWORD
```

Persistent Storage and Containers

Containers are ephemeral

A container's Writable Layer is deleted when the container is deleted

When a Pod is deleted, its container(s) is deleted from the Node

How can we persist data across a Pod's lifecycle?

Storage API object in Kubernetes

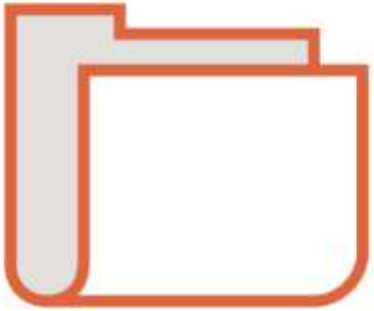
Volume

Persistent Volume

Persistent Volume Claim

Storage Class

Volume



Persistent storage deployed as part of the Pod spec

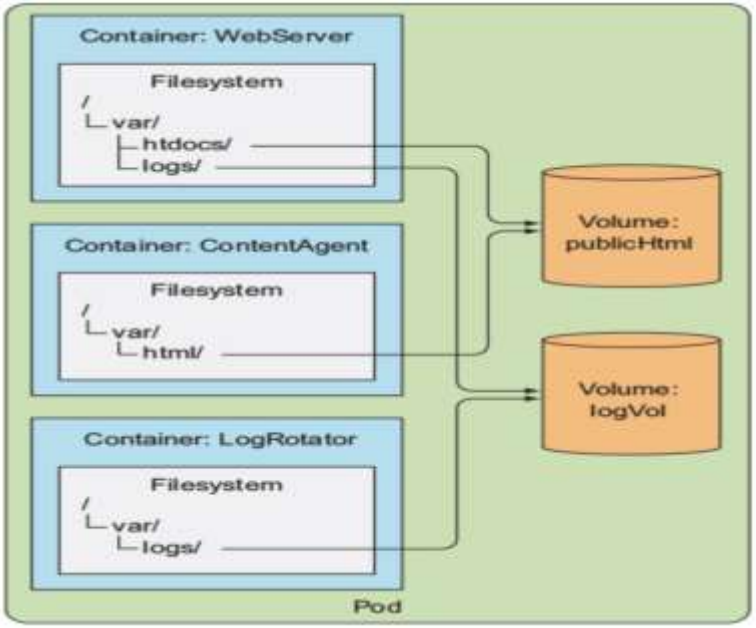
Implementation details for your storage

This can be challenging...

- Sharing code
- Same lifecycle as Pod

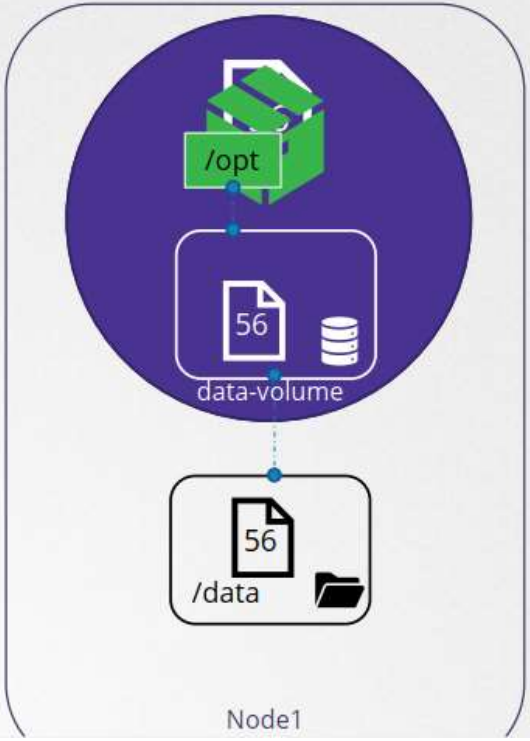
We can do better...

Volume


Container: WebServer
Filesystem
/
var/
htdocs/
logs/
Container: ContentAgent
Filesystem
/
var/
html/
Container: LogRotator
Filesystem
/
var/
logs/
Volume:
publicHtml
Volume:
logVol
Pod

Volume

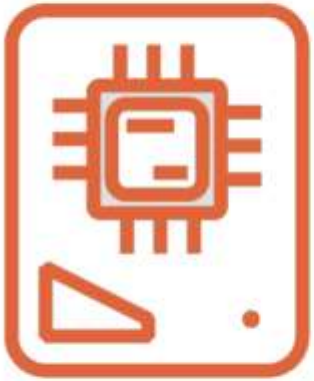
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: random-number-generator
spec:
  containers:
  - image: alpine
    name: alpine
    command: ["/bin/sh","-c"]
    args: ["shuf -i 0-100 -n 1 >> /opt/number.out;"]
    volumeMounts:
    - mountPath: /opt
      name: data-volume

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



Volume type

Temp	Local	Network
emptyDir	hostPath	GlusterFS gitRepo NFS iSCSI gcePersistentDisk AWS EBS azureDisk Fiber Channel Secret VshereVolume

Persistent Volume



Administrator defined storage in the Cluster

Implementation details for your storage

Lifecycle independent of the Pod

Managed by the Kubelet

- Maps the storage in the Node
- Exposes PV as a mount inside the container

Types of Persistent Volumes

Networked	Block	Cloud
NFS	Fibre Channel	awsElasticBlockStore
azureFile	iSCSI	azureDisk
		gcePersistentDisk

Persistent Volume Claim

A request for storage by a user

- Size
- Access Mode
- Storage Class

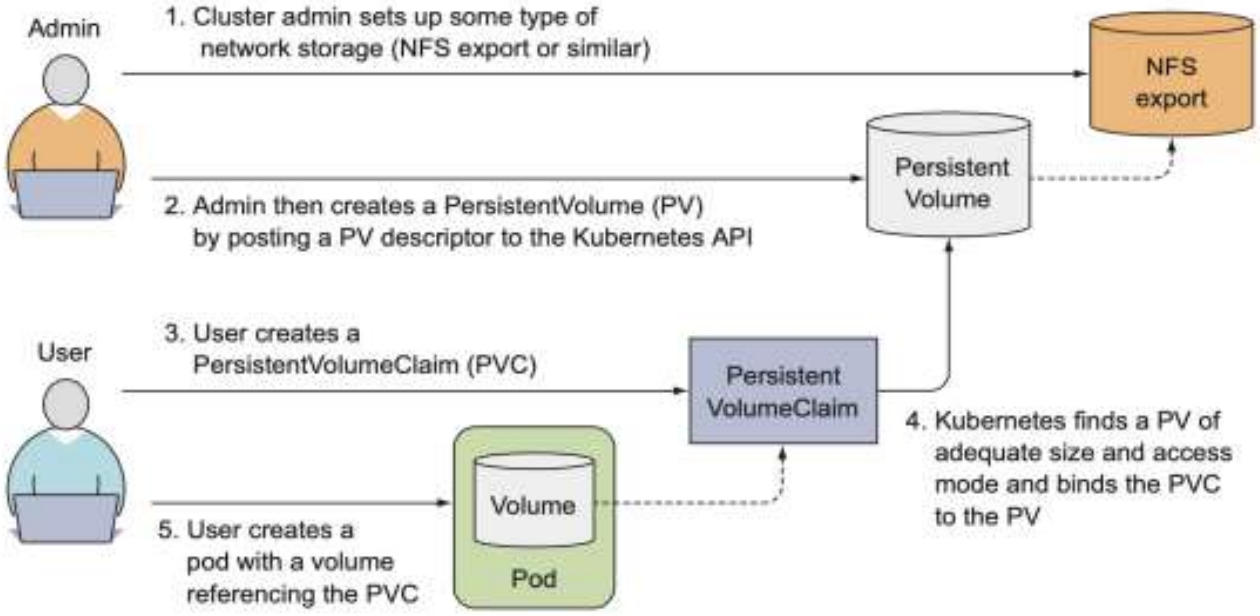
Enable portability of your application configurations

The Cluster will map a PVC to a PV

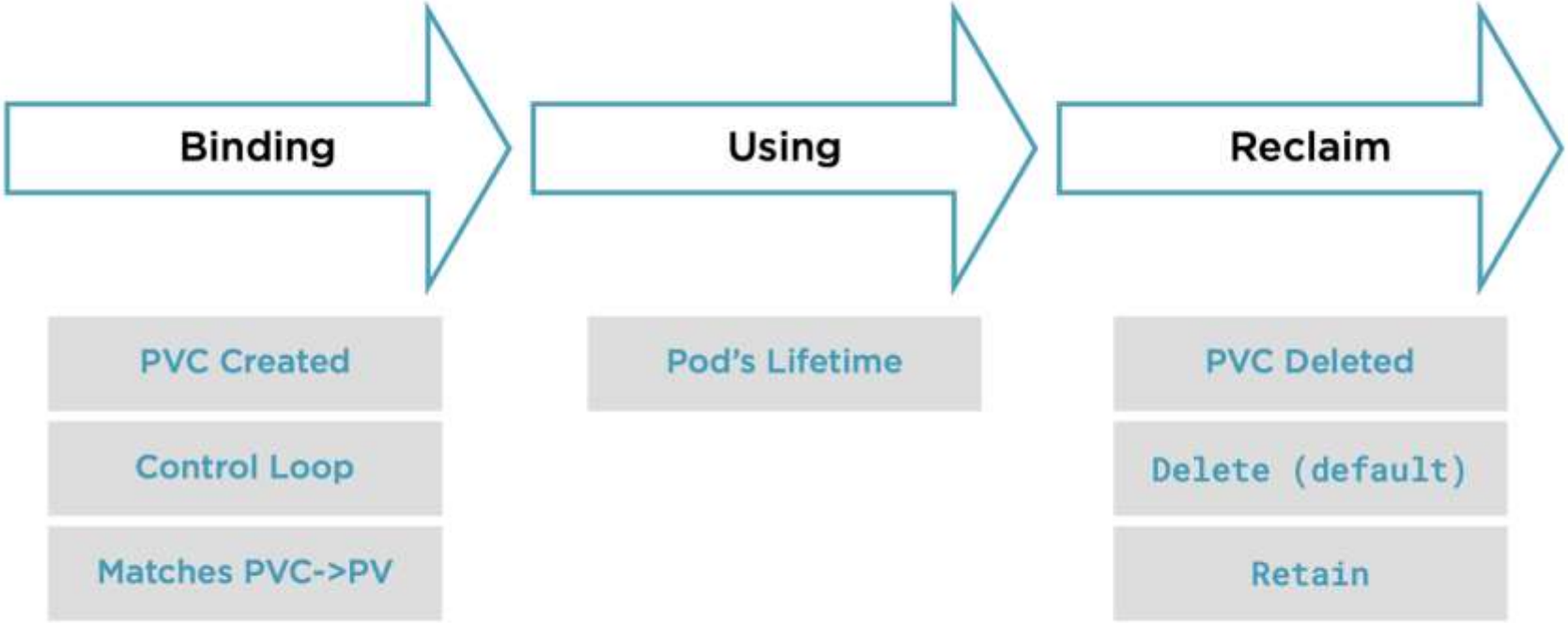
Access Modes

- RWO—ReadWriteOnce—Only a single node can mount the volume for reading and writing.
- ROX—ReadOnlyMany—Multiple nodes can mount the volume for reading.
- RWX—ReadWriteMany—Multiple nodes can mount the volume for both reading and writing

Static Provisioning Workflow



Storage Lifecycle



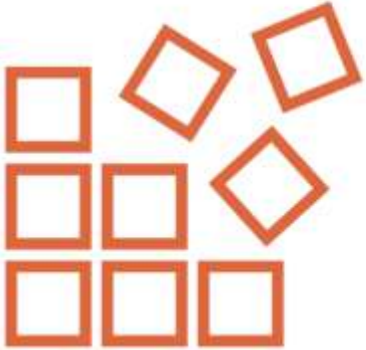
Define a Persistent Volume Claim

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-nfs-data
spec:
  accessModes:
    - ReadWriteMany
 resources:
    requests:
      storage: 10Gi
```

Using PVC in Pod

```
    command: ["/bin/bash","train.sh"]
    args: ["/detection-trainer/config/train.ini"]
    resources:
      requests:
        memory: 32Gi
        cpu: 4
        nvidia.com/gpu: 1
      limits:
        memory: 32Gi
        cpu: 4
        nvidia.com/gpu: 1
    env:
      - name: dvc_training_connection_string
        valueFrom:
          secretKeyRef:
            name: azure-secret
            key: dvc_training_connection_string
      - name: env
        value: "cloud"
      - name: AZURE_ACCOUNT_NAME
        value: "alprtraining"
      - name: AZURE_ACCOUNT_KEY
        valueFrom:
          secretKeyRef:
            name: azure-secret
            key: account_key
      - name: JOBID
        value: "plate-detection-trainer"
    volumeMounts:
      - mountPath: /dev/shm
        name: dshm
      - name: config
        mountPath: /detection-trainer/config
        readOnly: true
      - mountPath: "/mnt/data"
        name: dvc-volume
  restartPolicy: Never
  volumes:
    - name: dshm
      emptyDir:
        medium: Memory
    - name: config
      configMap:
        name: plate-detection-trainer-conf
    - name: dvc-volume
      persistentVolumeClaim:
        claimName: azure-managed-disk
```

Storage Class



Define tiers/classes of storage

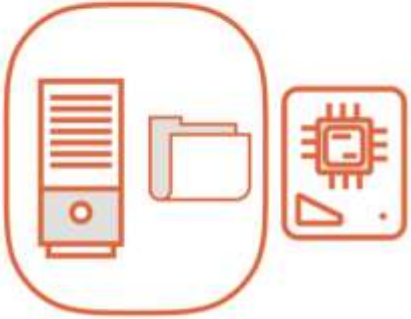
Enables Dynamic Provisioning

Define infrastructure specific parameters

Reclaim Policy

Dynamic Provisioning Workflow

Create a `StorageClass`	Create a `PersistentVolumeClaim`	Define `Volume` in Pod Spec
		Creates a `PersistentVolume`



Define Storage Class

```yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: managed-premium
parameters:
  kind: Managed
  storageaccounttype: Premium_LRS
provisioner: kubernetes.io/azure-disk
```

Dynamic provisioning

```yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: managed-premium
parameters:
  kind: Managed
  storageaccounttype: Premium_LRS
provisioner: kubernetes.io/azure-disk
```


Q
A