Lesson 2 : Provision Persistent Data Volumes

Kubernetes Persistent Storage

- Ephemeral storage
  - Cannot be shared
- Use persistent storage volume
- Two main objects:
  - Persistent Volume (PV)
  - Persistent Volume Claims (PVC)
- Cluster Admin configure PV
- Developers configure PVC

Provisioning storage

- Static provisioning:
  - Requires Cluster Admin
  - Manual process
  - Carry details of real storage
- Dynamic provisioning:
  - Uses storage classes
  - simplified storage management
  - preferred for its scalability and flexibility

- Storage classes describe:
  - type/tier of storage
  - quality of service level
  - backup policies

When to use Static Provisioning

- **Manual creation:** PVs are created manually by cluster administrator
- **Pre-allocated**: Static provisioning is helpful in scenarios with predictable storage usage, i.e., when parameters, such as the size of the volume, are predefined.
- **Existing volumes**: A storage volume already provisioned in your storage device can be attached to a Kubernetes cluster as static volume.
- **Shared volumes**: Static volumes can be used if the same volume has to be shared across different applications. This can be in the same project or between applications in different projects.

When to use Dynamic Provisioning?

- **Automated creation:** Pvs are automatically created by control plane based on user requests in PVC.

- **Flexibility**: Dynamic provisioning is helpful when you need flexibility, i.e., to meet changing usage demands. Storage size can also differ between applications, so a set static volume size will not be appropriate.

- **Infrastructure-as-Cod**e: Use cases where the volumes are created automatically using scripts or CI/CD tools would also need dynamic provisioning.

- **Optimization**: Helps optimize storage cost as the volumes are not pre-provisioned to utilize storage. This is especially helpful in cloud environments where you are charged for storage in a pay-as-you-go model.

Different features of Static and Dynamic storage provisioning

Features	Static Provisioning	Dynamic provisioning
Volume Sizing	Fixed sizing	Flexible sizing
Provisioning mode	Pre-provisioned	On-demand
Cloud Storage Cost	High	Low
DevOps integration	Complex	Easy

Persistent Volumes (PV) – Configured by Admin

- Volume plugins
- Have independent lifecycle
- Captures details of implementation of the storage: NFS, iSCSI or cloud provider specific
- Configuration :
  - Storage volume types, Volume Access Mode, Reclaim Policy, Volume Mode
- Not all storage is equal, it can differs in:
  - Cost
  - Performance
  - Reliability
  - Capacity

Storage volume types

- configMap volume: externalizes application configuration
- emptyDir: use for scratch data / temp data
- hostPath: mounts to file / directory on host node
- Glusterfs: free and open source scalable network filesystem
- iSCSI: block-level access to storage devices
- local: access local / direct-attached disks (das)
- NFS: access to remote exported folder on NFS servers
- Netapp FlexVolume : Netapp storage

Volume Access mode

Access mode	Abbreviation	Description
`ReadWriteOnce`	RWO	A single node mounts the volume as read/write.
`ReadOnlyMany`	ROX	Many nodes mount the volume as read-only.
`ReadWriteMany`	RWX	Many nodes mount the volume as read/write.

`ReadWriteOncePod`

FEATURE STATE: `Kubernetes v1.27 [beta]`

Reclaim Policy

Current reclaim policies:

- Retain -- manual reclamation
- Recycle (deprecated) -- basic scrub (rm -rf /thevolume/*)
- Delete -- associated storage asset such as AWS EBS or GCE PD volume is deleted

Currently, only NFS and HostPath support recycling. AWS EBS and GCE PD volumes support deletion.

Access Mode Support

Volume type	RWO	ROX	RWX
configMap	Yes	No	No
emptyDir	Yes	No	No
hostPath	Yes	No	No
iSCSI	Yes	Yes	No
local	Yes	No	No
NFS	Yes	Yes	Yes

Volume Modes

Filesystem (default)	Block
Have file system.	Do not have file system. Application own the storage and create the file system
Mounted into Pods into a directory	Presented as raw block device to Pod
User can access immediately	Performance benefits for applications
k8s will create the filesystem and mount to empty directory	Supported: CSI, FC, GCEPD, iSCSI, local volume, OpenStack Cinder, Vsphere Volume (vVol)

Block Volume support

Volume plug-in	Manually provisioned	Dynamically provisioned
AWS EBS	Yes	Yes
Azure disk	Yes	Yes
Cinder	Yes	Yes
Fibre channel	Yes	No
GCP	Yes	Yes
iSCSI	Yes	No
local	Yes	No
Red Hat OpenShift Data Foundation	Yes	Yes
VMware vSphere	Yes	Yes

Manually Creating a PV

```
apiVersion: v1
kind: PersistentVolume (1)
metadata:
  name: block-pv (2)
spec:
  capacity:
    storage: 10Gi (3)
  accessModes:
    - ReadWriteOnce (4)
  volumeMode: Block (5)
  persistentVolumeReclaimPolicy: Retain (6)
  fc: (7)
    targetWWNs: ["50060e801049cfd1"]
    lun: 0
    readOnly: false
```

1 PersistentVolume is the resource type for PVs.

2 Provide a name for the PV, which subsequent claims use to access the PV.

3 Specify the amount of storage that is allocated to this volume.

4 The storage device must support the access mode that the PV specifies.

5 The volumeMode attribute is optional for Filesystem volumes, but is required for Block volumes.

6 The persistentVolumeReclaimPolicy determines how the cluster handles the PV when the PVC is deleted. Valid options are Retain or Delete.

7 The remaining attributes are specific to the storage type. In this example, the fc object specifies the Fiber Channel storage type attributes.

```
[user@host]$ oc create -f my-fc-volume.yaml
```

oc describe pv Command

```
kubectl describe pv task-pv-volume
Name:            task-pv-volume
Labels:          type=local
Annotations:     <none>
Finalizers:      [kubernetes.io/pv-protection]
StorageClass:    standard
Status:          Terminating
Claim:
Reclaim Policy:  Delete
Access Modes:    RWO
Capacity:        1Gi
Message:
Source:
    Type:          HostPath (bare host directory volume)
    Path:          /tmp/data
    HostPathType:
Events:            <none>
```

Persistent Volume Claim (PVC)

- Request for storage by a user / developer
- Pods consume node resources; PVCs consume PV resources
- PVC declares what application needs, K8s provide best-effort basis
  - May specify storage type/tier (should be done in PV), access mode or reclaim policy
- Minimal storage characteristics
  - Request storage size
  - Specifies access modes
- Lifecycle tied to namespace not pod
- Multiple pods connect to same PVC

Manually Creating a PVC using oc set volumes command

- Add PVC to deployment

```
[user@host ~]$ oc set volumes deployment/example-application \ (1)
--add \ (2)
--name example-pv-storage \ (3)
--type persistentVolumeClaim \ (4)
--claim-mode rwo \ (5)
--claim-size 15Gi \ (6)
--mount-path /var/lib/example-app \ (7)
--claim-name example-pv-claim (8)
```

- Updating volumes

```
$ oc set volume <object_type>/<name> --add --overwrite [options]
```

- Remove volumes

```
$ oc set volume <object_type>/<name> --remove [options]
```

Manually Creating a PVC using YAML manifest

```
---
apiVersion: v1
kind: PersistentVolumeClaim (1)
metadata:
  name: example-pv-claim (2)
  labels:
    app: example-application
spec:
  accessModes:
    - ReadWriteOnce (3)
  resources:
    requests:
      storage: 15Gi (4)
```

1 PersistentVolumeClaim is the resource type for a PVC.

2 Use this name in the claimName field of the persistentVolumeClaim element in the volumes section of a deployment manifest.

3 Specify the access mode that this PVC requests. The storage class provisioner must provide this access mode. If persistent volumes are created statically, then an eligible persistent volume must provide this access mode.

4 The storage class creates a persistent volume that matches this size request. If persistent volumes are created statically, then an eligible persistent volume must be at least the requested size.

```
[user@host ~]$ oc create -f pvc_file_name.yaml
```

Verify PVC and web console

```
[user@host ~]$ oc get pvc
NAME         STATUS   VOLUME          CAPACITY   ACCESS MODES   STORAGECLASS   ...
db-pod-pvc   Bound    pvc-13...ca45   1Gi        RWO            nfs-storage    ...
```

To create a persistent volume claim from the web console, click the **Storage** > **PersistentVolumesClaims** menu.

Click **Create PersistentVolumeClaim** and complete the form by adding the name, the storage class, the size, the access mode, and the volume mode.

Kubernetes Dynamic Provisioning

- Cluster admin must statistically provision storage types
- For on demand or dynamic provisioning, uses **StorageClass** object
- Minimal configuration:
  - access mode
  - storage class
  - size
- OpenShift master node watches for new PVC and binds to appropriate PV
- Example: cluster with many 50Gi PV
  - Pod attempt with 100Gi claim – unbound
  - Later PV with 100Gi added, then Pod bound to it

The commands

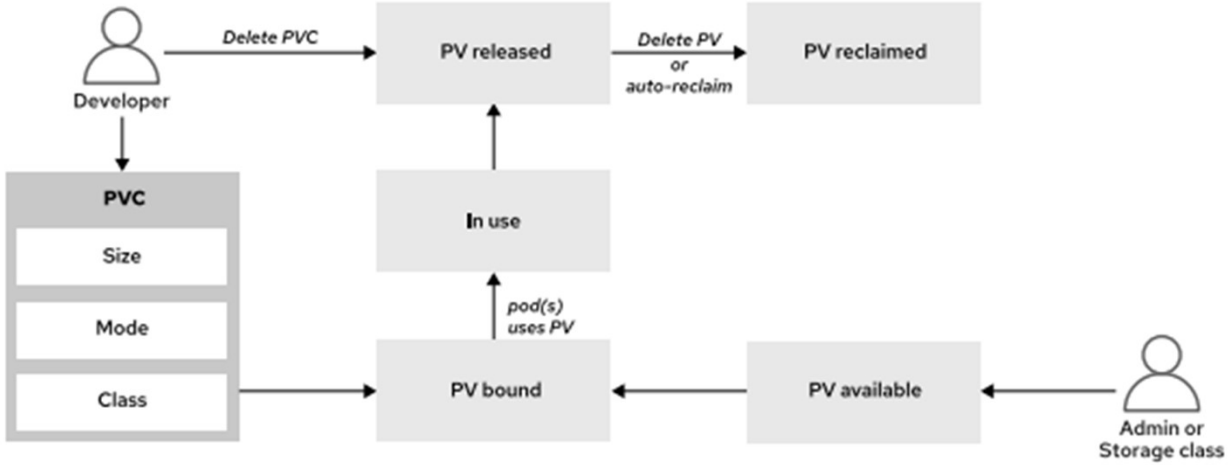
- Use `oc get storageclass` to view the storage classes that the cluster provides.

```
[user@host ~]$ oc get storageclass
NAME                    PROVISIONER                                    ...
nfs-storage (default)   k8s-sigs.io/nfs-subdir-external-provisioner   ...
lvms-vg1             topolvm.io                                ...
```

- The following `oc set volume` command example uses the `claim-class` option to specify a dynamically provisioned PV.

```
[user@host ~]$ oc set volumes deployment/example-application \
--add --name example-pv-storage --type pvc --claim-class nfs-storage \
--claim-mode rwo --claim-size 15Gi --mount-path /var/lib/example-app \
--claim-name example-pv-claim
```

PV and PVC Lifecycles



Status

Available

After a PV is created, it becomes *available* for any PVC to use in the cluster in any namespace.

Bound

A PV that is *bound* to a PVC is also bound to the same namespace as the PVC, and no other PVC can use it.

In Use

You can delete a PVC if no pods actively use it. The *Storage Object in Use Protection* feature prevents the removal of bound PVs and PVCs that pods are actively using. Such removal can result in data loss. Storage Object in Use Protection is enabled by default.

If a user deletes a PVC that a pod actively uses, then the PVC is not removed immediately. PVC removal is postponed until no pods actively use the PVC. Also, if a cluster administrator deletes a PV that is bound to a PVC, then the PV is not removed immediately. PV removal is postponed until the PV is no longer bound to a PVC.

Released

After the developer deletes the PVC that is bound to a PV, the PV is *released*, and the storage that the PV used can be reclaimed.

Reclaimed

The reclaim policy of a persistent volume tells the cluster what to do with the volume after it is released. A volume's reclaim policy can be Retain or Delete.

Deleting a Persistent Volume Claim

- **Deleting a Persistent Volume Claim**

  To delete a volume, use the `oc delete` command to delete the PVC. The storage class reclaims the volume after removing the PVC.

```
[user@host ~]$ oc delete pvc/example-pvc-storage
```

- **Volume Reclaim Policy**

Policy	Description
Retain	Enables manual reclamation of the resource for those volume plug-ins that support it.
Delete	Deletes both the `PersistentVolume` object from OpenShift Container Platform and the associated storage asset in external infrastructure.

Guided Exercise: Provision Persistent Data Volumes

You should be able to:

- Deploy a MySQL database with persistent storage from a PVC.
- Identify the PV that backs the application.
- Identify the storage provisioner that created the PV

Lab: Deploy Managed and Networked Applications on Kubernetes

You should be able to:

- Deploy a MySQL database from a container image.
- Deploy a web application from a container image.
- Configure environment variables for a deployment.
- Expose the web application for external access.

Quiz 1

Which method for creating container images is recommended by the containers community?

a) Run commands inside a basic OS container, commit the container, and save or export it as a new container image.

b) Run commands from a Dockerfile and push the generated container image to an image registry.

c) Create the container image layers manually from tar files.

d) Run the podman build command to process a container image description in YAML format.

Quiz 1

Which method for creating container images is recommended by the containers community?

a) Run commands inside a basic OS container, commit the container, and save or export it as a new container image.

b) Run commands from a Dockerfile and push the generated container image to an image registry.

c) Create the container image layers manually from tar files.

d) Run the podman build command to process a container image description in YAML format.

Quiz 2

What are two advantages of using the standalone S2I process as an alternative to Dockerfiles? (Choose two.)

a) Requires no additional tools apart from a basic Podman setup.

b) Creates smaller container images, having fewer layers.

c) Reuses high-quality builder images.

d) Automatically updates the child image as the parent image changes (for example, with security fixes).

Quiz 2

What are two advantages of using the standalone S2I process as an alternative to Dockerfiles? (Choose two.)

a) Requires no additional tools apart from a basic Podman setup.

b) Creates smaller container images, having fewer layers.

c) Reuses high-quality builder images.

d) Automatically updates the child image as the parent image changes (for example, with security fixes).

Quiz 3

What are two typical scenarios for creating a Dockerfile to build a child image from an existing image? (Choose two.)

a) Adding new runtime libraries.

b) Setting constraints to a container's access to the host machine's CPU.

c) Adding internal libraries to be shared as a single image layer by multiple container images for different applications.

d) Creates images compatible with OpenShift, unlike container images created from Docker tools.

Quiz 3

What are two typical scenarios for creating a Dockerfile to build a child image from an existing image? (Choose two.)

a) Adding new runtime libraries.

b) Setting constraints to a container's access to the host machine's CPU.

c) Adding internal libraries to be shared as a single image layer by multiple container images for different applications.

a) Creates images compatible with OpenShift, unlike container images created from Docker tools.

Layering Image

- Applicable to other INSTRUCTIONs

```
LABEL version="2.0" \
      description="This is an example container image" \
      creationDate="01-09-2017"
```

```
ENV MYSQL_ROOT_PASSWORD="my_password" \
    MYSQL_DATABASE "my_database"
```

```
EXPOSE 8080, 9090, 9191
```

Chapter Summary

In this chapter, you learned:

Dockerfile contains instructions that specify how to construct a container image.

The Source-to-Image (S2I) process provides an alternative to Dockerfiles. S2I implements a standardized container image build process for common technologies from application source code. This allows developers to focus on application development and not Dockerfile development.

Container images provided by Red Hat Container Catalog or Quay.io are a good starting point for creating custom images for a specific language or technology.

Building an image from a Dockerfile using a three-step process

You should be able to:

- Create network policies to control communication between pods.
- Verify ingress traffic is limited to pods.

You should be able to:

- Modify a secret to add htpasswd entries for new users.
- Configure a new project with role-based access controls and resource quotas.
- Use an OperatorHub operator to deploy a database.
- Create a deployment, service, and route for a web application.
- Troubleshoot an application using events and logs.

In this chapter, you learned:

The OpenShift web console provides a GUI for visualizing and managing OpenShift resources.

Some resources feature a specialized page that makes creating and editing resources more convenient than writing YAML by hand, such as the Edit Key/Value Secret editor, which automatically handles Base64 encoding and decoding.

You can install partner and community operators from the embedded OperatorHub page.

Cluster-wide metrics such as CPU, memory, and storage usage are displayed on the Dashboards page.

Project Details pages display metrics specific to the project, such as the top ten memory consumers by pod and the current resource quota usage.