# Lab 7 – Configure Persistent Storage for Kubernetes

## Introduction

In this lab, you will learn how to create a persistent volume, persistent volume claim and also reclaim the volume policy and create the storage classes.

Containers are ephemeral, meaning the container file system only lives as long as the container does. Volumes are simplest way to achieve data persistance. In kubernetes, a more flexible and powerful model is available.

This model is based on the following abstractions:

- **PersistentVolume:** it models shared storage that has been provisioned by the cluster administrator. It is a resource in the cluster just like a node is a cluster resource. Persistent volumes are like standard volumes, but having a lifecycle independent of any individual pod. Also they hide to the users the details of the implementation of the storage, e.g. NFS, iSCSI, or other cloud storage systems.
- **PersistentVolumeClaim:** it is a request for storage by a user. It is similar to a pod. Pods consume node resources and persistent volume claims consume persistent volume objects. As pods can request specific levels of resources like cpu and memory, volume claimes claims can request the access modes like read-write or read-only and stoarage capacity.

Kubernetes provides two different ways to provisioning storage:

- **Manual Provisioning:** the cluster administrator has to manually make calls to the storage infrastructure to create persisten volumes and then users need to create volume claims to consume storage volumes.
- **Dynamic Provisioning:** storage volumes are automatically created on-demand when users claim for storage avoiding the cluster administrator to pre-provision storage.

## 1. Local Persistent Volumes

**1.1** Make sure you, logged in as ***"root"*** user on **pod0-spare.origin.com** host

Copy

```
ssh root@pod0-master.origin.com
```

**1.2** Create a persistent volume as “**local-persistent-volume-recycle.yaml**” file:

Copy

```
cat > local-persistent-volume-recycle.yaml <<EOF

kind: PersistentVolume

apiVersion: v1

metadata:

  name: local-pv

  labels:

    type: local

spec:

  storageClassName: ""

  capacity:

    storage: 2Gi

  accessModes:

    - ReadWriteOnce

  hostPath:

    path: "/data"

  persistentVolumeReclaimPolicy: Recycle

EOF
```

The configuration file specifies that the volume is at /**data** on the the cluster’s node. The volume type is **hostPath** meaning the volume is local to the host node. The configuration also specifies a size of 2GB and the access mode of **ReadWriteOnce**, meanings the volume can be mounted as read write by a single pod at time.

The reclaim policy is **Recycle** meaning the volume can be used many times. It defines the Storage Class name manual for the persistent volume, which will be used to bind a claim to this volume.

**1.3** Run the below command to create local persistent volume.

Copy

```
kubectl create -f local-persistent-volume-recycle.yaml
```

**Output:**

```
persistentvolume "local-pv" created
```

**1.4** Verify the persistent volume status

Copy

```
kubectl get pv
```

**Output:**

```
NAME      CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM
STORAGECLASS   REASON    AGE

local-pv     2Gi        RWO            Recycle          Available
37s
```

**1.5** Create a persistent volume claim as "**volume-claim.yaml**" file:

Copy

```
cat > volumeclaim-pvc.yaml <<EOF

kind: PersistentVolumeClaim

apiVersion: v1

metadata:

  name: volumeclaim-pvc
```

```
spec:

  storageClassName: ""

  accessModes:

    - ReadWriteOnce

  resources:

    requests:

      storage: 1Gi

EOF
```

**Note:** The claim is for 1GB of space where the the volume is 2GB. The claim will bound any volume meeting the minimum requirements specified into the claim definition.

**1.6** Create the persistent volume claim

Copy

```
kubectl create -f volumeclaim-pvc.yaml
```

**Output:**

```
persistentvolumeclaim "volumeclaim-pvc" created
```

**1.7** Verify that the persistent volume status is in “**bound**”

Copy

```
kubectl get pv
```

**Output:**

```
NAME      CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS    CLAIM
STORAGECLASS   REASON    AGE
```

```
localpv   2Gi        RWO            Recycle          Bound     default
/volumeclaim-pvc                          4m
```

**1.8** Verify that the persistent volume claim status is in **Bound**

Copy

```
kubectl get pvc
```

**Output:**

```
NAME             STATUS    VOLUME    CAPACITY   ACCESS MODES   STORAG
ECLASS   AGE

volumeclaim-pvc   Bound     localpv   2Gi        RWO
9s
```

**1.9** Create a **nginx-pod-pvc.yaml** configuration file for a nginx pod using the above claim for its html content directory

Copy

```
cat > nginx-pod-pvc.yaml <<EOF

---

kind: Pod

apiVersion: v1

metadata:

  name: nginx

  namespace: default

  labels:

spec:
```

```
  containers:

    - name: nginx

      image: nginx:latest

      ports:

        - containerPort: 80

          name: "http-server"

      volumeMounts:

      - mountPath: "/usr/share/nginx/html"

        name: html



  volumes:

    - name: html

      persistentVolumeClaim:

       claimName: volumeclaim-pvc

EOF
```

**Note:** The pod configuration file specifies a persistent volume claim, but it does not specify a persistent volume. From the pod point of view, the claim is the volume. Please note that a claim must exist in the same namespace as the pod using the claim.

**1.10** Create the nginx pod from presitence Volume claim

Copy

```
kubectl create -f nginx-pod-pvc.yaml
```

**Output:**

```
pod "nginx" created
```

**1.11** Accessing the nginx will return ***403 Forbidden*** since there are no html files to serve in the data volume

Copy

```
kubectl get pod
```

**Output:**

```
NAME      READY     STATUS    RESTARTS   AGE

nginx     1/1       Running   0          1m
```

**Note:** Wait till pod changes to **running** state.

**1.12** Verify the nginx pod status and get pod IP's

Copy

```
kubectl get pod nginx -o yaml | grep IP
```

**Sample output:**

```
  hostIP: 10.1.64.246

  podIP: 10.244.1.36
```

**1.13** Capture the pod ip's from the nginx server

Copy

```
pod_ip=`kubectl describe  pod nginx | grep IP | cut -d " " -f 12`

echo $pod_ip
```

**Sample output:**

```
10.244.1.36
```

**1.14** Try to access the nginx application via pod_ip , but there are no html files to serve in the data volume

Copy

```
curl http://$pod_ip:80
```

**Output:**

```
403 Forbidden
```

**1.15** Let's populate the data volume by switching to worker node **pod0-node** .

Copy

```
ssh root@pod0-master.origin.com
```

Copy

```
echo "Welcome to $(hostname)" > /data/index.html
```

**1.16** Exit from **pod0-master.origin.com** host:

Copy

```
exit
```

**Note:**Run the below commands on **pod0-master.origin.com**.

**1.17** Now try again to access the nginx application :

Copy

```
curl http://$pod_ip:80
```

**Output:**

```
Welcome to pod40-node.onecloud.com
```

**1.18** To test the persistence of the volume and related claim, delete the pod and recreate it

Copy

```
kubectl delete pod nginx
```

**Output:**

```
pod "nginx" deleted
```

**1.19** Create a nginx pod for persistence volume claim

Copy

```
kubectl create -f nginx-pod-pvc.yaml
```

**Output:**

```
pod "nginx" created
```

**1.20** Verify the nginx pod status is in "**running**":

Copy

```
kubectl get pod nginx
```

**Output:**

```
NAME      READY     STATUS    RESTARTS   AGE

nginx     1/1       Running   0          1m
```

**Note:** Wait till pod changes to **running** state.

**1.21** Locate the IP of the new nginx pod and try to access it

Copy

```
pod_ip=`kubectl describe  pod nginx | grep IP | cut -d " " -f 12`

echo $pod_ip
```

**Sample output:**

```
10.244.1.37
```

**1.22** Access the nginx application

Copy

```
curl http://$pod_ip
```

**Output:**

```
Welcome to pod40-node.onecloud.com
```

# 2. Volume state

When a pod claims for a volume, the cluster inspects the claim to find the volume meeting claim requirements and mounts that volume for the pod. Once a pod has a claim and that claim is bound, the bound volume belongs to the pod.

A volume will be in one of the following state:

- **Available**: a volume that is not yet bound to a claim
- **Bound**: the volume is bound to a claim
- **Released**: the claim has been deleted, but the volume is not yet available
- **Failed**: the volume has failed

The volume is considered released when the claim is deleted, but it is not yet available for another claim. Once the volume becomes available again then it can bound to another other claim.

**2.1** Delete the pod

Copy

```
kubectl delete pod nginx
```

**Output:**

```
pod "nginx" deleted
```

**2.2** Delete the persistent volume claim

Copy

```
kubectl delete pvc volumeclaim-pvc
```

**Output:**

```
persistentvolumeclaim "volumeclaim-pvc" deleted
```

**2.3** Verify the status of the volume

Copy

```
kubectl get pv local-pv
```

**Output:**

```
NAME      CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS     CLAIM
STORAGECLASS   REASON    AGE

localpv   2Gi        RWO            Recycle          Released   defaul
t/volumeclaim-pvc                         9m
```

# 3. Volume Reclaim Policy

When deleting a claim, the volume becomes available to other claims only when the volume claim policy is set to Recycle. Volume claim policies currently supported are:

A volume will be in one of the following state:

- **Retain**: the content of the volume still exists when the volume is unbound and the volume is released
- **Recycle**: the content of the volume is deleted when the volume is unbound and the volume is available
- **Delete**: the content and the volume are deleted when the volume is unbound.

Please note that, currently, only NFS and HostPath support recycling.

When the policy is set to Retain the volume is released but it is not yet available for another claim because the previous claimant's data are still on the volume.

**3.1** Define a persistent volume using **local-persistent-volume-retain.yaml** configuration file

Copy

```
cat > local-persistent-volume-retain.yaml <<EOF

kind: PersistentVolume

apiVersion: v1

metadata:

  name: local-retain

  labels:

    type: local

spec:

  storageClassName: ""

  capacity:

    storage: 2Gi

  accessModes:

    - ReadWriteOnce

  hostPath:
```

```
    path: "/data"

  persistentVolumeReclaimPolicy: Retain

EOF
```

**3.2** Create the persistent volume and the claim

Copy

```
kubectl create -f local-persistent-volume-retain.yaml

kubectl create -f volumeclaim-pvc.yaml
```

**Output:**

```
persistentvolume "local-retain" created

persistentvolumeclaim "volumeclaim-pvc" created
```

**3.3** Create the nginx pod for persistent volume claim

Copy

```
kubectl create -f nginx-pod-pvc.yaml
```

**Output:**

```
pod "nginx" created
```

**3.4** Verify the pod status is in “running”

Copy

```
kubectl get pods
```

**Output:**

```
NAME      READY     STATUS    RESTARTS   AGE

nginx     1/1       Running   0          2m
```

**Note:** Wait till pod changes to **Running** status, before proceeding to next step .

**3.5** Login to the pod using the claim and create some data on the volume

Copy

```
kubectl exec -it nginx bash
```

**Output:**

```
root@nginx:/#
```

**3.6** Add data to index.html file

Copy

```
echo "Hello World" > /usr/share/nginx/html/index.html
```

**3.7** Exit from the nginx server

Copy

```
exit
```

**3.8** Verify the status of the volume

Copy

```
kubectl get pv
```

**Output:**

```
NAME           CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS
CLAIM                     STORAGECLASS   REASON    AGE
```

```
local-pv       2Gi        RWO            Recycle          Available
10m

local-retain   2Gi        RWO            Retain           Bound
default/volumeclaim-pvc                          3m
```

We see the volume remain in the released status and not becomes available since the reclaim policy is set to Retain.

**3.9** Now login to the worker node and check data are still there.

An administrator can manually reclaim the volume by deleteting the volume and creating a another one.

# 4. Manual volumes provisioning

In this section we're going to use a Network File System storage backend for manual provisioning of shared volumes. Main limit of local storage for container volumes is that storage area is tied to the host where it resides. If kubernetes moves the pod from another host, the moved pod is no more to access the data since local storage is not shared between multiple hosts of the cluster. To achieve a more useful storage backend we need to leverage on a shared storage technology like NFS.

**4.1** Create a NFS share directory on master node as we do not have a dedicated NFS server for this demo.

Copy

```
mkdir /mnt/nfs
```

**4.2** Add entry to ***/etc/exports*** to describe NFS share

Copy

```
cat >> /etc/exports <<EOF

/mnt/nfs *(rw,async,no_root_squash)

EOF
```

**4.3** Enable and start NFS server daemon

Copy

```
systemctl enable nfs-server

systemctl start nfs-server
```

**Output:**

```
Created symlink from /etc/systemd/system/multi-user.target.wants/nfs-s
erver.service to /usr/lib/systemd/system/nfs-server.service.
```

**4.4** Verify the NFS share is available

Copy

```
exportfs
```

**Output:**

```
/mnt/nfs        <world>
```

**4.5** Define a persistent volume as in the **nfs-persistent-volume.yaml** configuration file

Copy

```
cat > nfs-persistent-volume.yaml <<EOF

apiVersion: v1

kind: PersistentVolume

metadata:

  name: nfs-volume

spec:

  storageClassName: ""

  capacity:
```

```
    storage: 1Gi
  accessModes:
  - ReadWriteOnce
  nfs:
    path: "/mnt/nfs"
    server: pod0-master
  persistentVolumeReclaimPolicy: Recycle
EOF
```

**4.6** Create the nfs persistent volume

Copy

```
kubectl create -f nfs-persistent-volume.yaml
```

**Output:**

```
persistentvolume "nfs-volume" created
```

**4.7** Verify the persistent volume nfs-volume status

Copy

```
kubectl get pv nfs-volume -o wide
```

**Output:**

```
NAME          CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS
CLAIM                    STORAGECLASS   REASON    AGE

nfs-volume    1Gi        RWO            Recycle          Available
7s
```

**4.8** Create a persistent volume claim using **volume-claim.yaml**

Copy

```
cat > volume-claim.yaml <<EOF

kind: PersistentVolumeClaim

apiVersion: v1

metadata:

  name: volume-claim

spec:

  storageClassName: ""

  accessModes:

    - ReadWriteOnce

  resources:

    requests:

      storage: 1Gi

EOF
```

Copy

```
kubectl create -f volume-claim.yaml
```

**Output:**

```
persistentvolumeclaim "volume-claim" created
```

**4.9** Verify volume-claim status is in “bound” status

Copy

```
kubectl get pvc
```

**Output:**

```
NAME              STATUS        VOLUME       CAPACITY   ACCESS MODES
STORAGECLASS   AGE

volume-claim      Bound         nfs-volume   1Gi        RWO
54m

volumeclaim-pvc   Bound         localpv      2Gi        RWO
59m
```

**4.10** Verify persistent volume-claim status

Copy

```
kubectl get pv
```

**Output:**

```
NAME           CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS
CLAIM                     STORAGECLASS   REASON    AGE

local-pv       2Gi        RWO            Recycle          Available
13m

local-retain   2Gi        RWO            Retain           Bound
default/volumeclaim-pvc                                  6m

nfs-volume     1Gi        RWO            Recycle          Bound
default/volume-claim                                     1m
```

**4.11** Now we are going to create more nginx pods using the same claim.

Create the **nginx-pvc-template.yaml** template for a nginx application having the html content folder placed on the shared storage.

Copy

```
cat > nginx-pvc-template.yaml <<EOF
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  generation: 1
  labels:
    run: nginx
  name: nginx-pvc
spec:
  replicas: 3
  selector:
    matchLabels:
      run: nginx
  strategy:
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 1
    type: RollingUpdate
  template:
    metadata:
      labels:
```

```
        run: nginx
    spec:
      containers:
      - image: nginx:latest
        imagePullPolicy: IfNotPresent
        name: nginx
        ports:
        - containerPort: 80
          protocol: TCP
          name: "http-server"
        volumeMounts:
        - mountPath: "/usr/share/nginx/html"
          name: html
      volumes:
      - name: html
        persistentVolumeClaim:
          claimName: volume-claim
      dnsPolicy: ClusterFirst
      restartPolicy: Always
EOF
```

The template above defines a nginx application based on a nginx deploy of 3 replicas. The nginx application requires a shared volume for its html content. The application does not have to deal with complexity of setup and admin an NFS share.

**4.12** Deploy the application

Copy

```
kubectl create -f nginx-pvc-template.yaml
```

**Output:**

```
deployment.extensions "nginx-pvc" created
```

**4.13** Check all pods are up and running

Copy

```
kubectl get pods -o wide
```

**Sample output:**

```
NAME                         READY     STATUS    RESTARTS   AGE
IP           NODE

nginx                        1/1       Running   0          7m
10.244.1.5   pod38-node.onecloud.com

nginx-pvc-5846b844f5-2tmnk   1/1       Running   0          1m
10.244.1.6   pod38-node.onecloud.com

nginx-pvc-5846b844f5-cnq4c   1/1       Running   0          1m
10.244.0.3   pod38-node.onecloud.com

nginx-pvc-5846b844f5-dpfsr   1/1       Running   0          1m
10.244.1.7   pod38-node.onecloud.com
```

**Note:** Wait till the pod status changes to **"running"** status. **Re-run** the above command to verify of it.

**4.14** Login to one of these pods and add html content

Copy

```
kubectl get pods -o wide | grep "nginx-pvc" | awk '{print $1}'
```

**Sample output:**

```
nginx-pvc-5846b844f5-2tmnk

nginx-pvc-5846b844f5-cnq4c

nginx-pvc-5846b844f5-dpfsr
```

**4.15** Capture the pod_name from the created nginx persistent volume claim

Copy

```
pod_name=`kubectl get pods -o wide | grep "nginx-pvc" | awk '{print $1
}' | head -1`

 echo $pod_name
```

**Sample output:**

```
nginx-pvc-5846b844f5-2tmnk
```

**4.16** Try to access the nginx persistent volume claim server and html content to **index.html**

Copy

```
kubectl exec -it $pod_name bash
```

**Output:**

```
root@nginx-pvc-5846b844f5-2tmnk:/#
```

Copy

```
echo 'Hello from NFS!' > /usr/share/nginx/html/index.html
```

Copy

```
exit
```

**4.17** Since all three pods mount the same shared folder on the NFS, the just created html content is placed on the NFS share and it is accessible from any of the three pods

Copy

```
kubectl get pod  -o wide | grep nginx-pvc | awk '{print $6}'
```

**Sample output:**

```
10.32.0.3

10.44.0.1

10.44.0.2
```

**4.18** Capture the pod ip from the nginx pvc

Copy

```
pod_ip=`kubectl get pod  -o wide | grep nginx-pvc | awk '{print $6}' |
head -1`

echo $pod_ip
```

**Sample output:**

```
10.32.0.3
```

Copy

```
curl http://$pod_ip
```

**Output:**

```
Hello from NFS!
```

# 5. Cleanup

**5.1** Delete all the pods

Copy

```
kubectl delete pods --all
```

**Sample output:**

```
pod "nginx" deleted

pod "nginx-pvc-5846b844f5-2tmnk" deleted

pod "nginx-pvc-5846b844f5-cnq4c" deleted

pod "nginx-pvc-5846b844f5-dpfsr" deleted
```