PV&PC

Friday, January 7, 2022 10:30 AM

How to use Persistent Volume and Persistent Claims | Kubernetes

Working with kubernetes is always fun as well as challenging. The more you dive deep into the kubernetes ecosystem the more you learn.
It always bugged me when I started working with kubernetes that - How can I retain the data after the end of pod life cycle?

Answer is -

> Kubernetes Persistent Volume and Persistent claims help you to retain the data of the pod even after the end of the pod life cycle

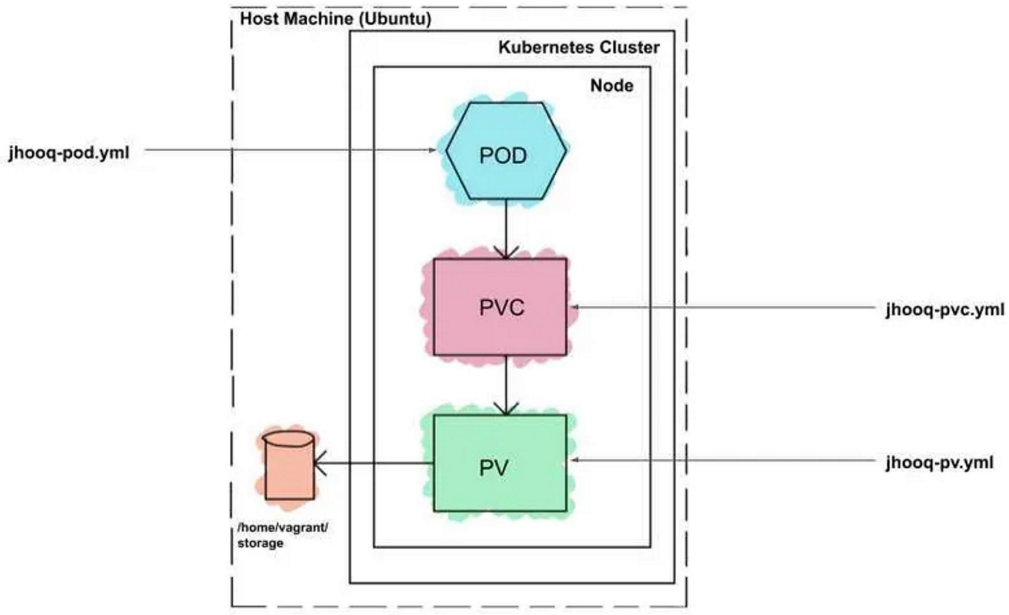
What problems does it solve?

Containers running inside the pod can not share the files with each other.

1. All the files inside the container are temporary which means if you terminate the container you are going to lose all your files.
2. Secondly if in any case, your container crashes then there is no way to recover files.

Kuberenetes provides volume plugin as **Persistent Volume** to address the above problems.

The lifecycle of these volumes are independent of the lifecycle of pods.
So if PODs are terminated then volumes are unmounted and detached keeping the data intact.



What is Persistent Volume(PV)?

In simple terms, it's storage available within your Kubernetes cluster. This storage can be provisioned by you or Kubernetes administrator.

It's basically a directory with some data in it and all the containers running inside the pods can access it. But Persistent Volumes are independent of the POD life cycle.

So if PODs live or die, persistent volume does get affected and it can be reused by some other PODs.

Kubernetes provides many volume plugins based on the cloud service provider you are using -

awsElasticBlockStore, azureDisk, azureFile, cephfs, cinder, configMap, csi, downwardAPI, emptyDir, fc (fibre channel), flexVolume, flocker, gcePersistentDisk, gitRepo (deprecated), glusterfs, hostPath, iscsi, local, nfs, persistentVolumeClaim, projected, portworxVolume, quobyte, rbd, scaleIO, secret, storageos, vsphereVolume

How can you Create persistent volume?

There are some prerequisites before you create your persistent volume

Step 1- Prerequisites

1. **Kubernetes Cluster**: - You should have Kubernetes cluster up and running (**If you do not know "How to setup then please refer to** this article)
2. **Docker Image/container**: - A docker image that can be deployed as Kubernetes deployment. (I am going to use Spring Boot Docker image. If you do not have then please refer on **how to create docker image you can follow** this article)
3. **Directory for persistent Volume storage:-** Create one directory .i.e. **/home/vagrant/storage** inside your Linux machine for persistent volume. You can keep the directory name and path as per your need

Step 2 - Create a persistent volume configuration (jhooq-pv.yml)
Its fairly easy to create it. Refer to the following configuration which you can customize with your requirements -


```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: jhooq-demo-pv
spec:
  capacity:
    storage: 1Gi
  volumeMode: Filesystem
  accessModes:
  - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  storageClassName: local-storage
  local:
    path: /home/vagrant/storage
  nodeAffinity:
    required:
      nodeSelectorTerms:
      - matchExpressions:
        - key: kubernetes.io/hostname
          operator: In
          values:
          - node1
```

apiVersion: v1
kind: PersistentVolume
metadata:
 name: jhooq-demo-pv
spec:
 capacity:
  storage: 1Gi
 volumeMode: Filesystem
 accessModes:
 - ReadWriteOnce
 persistentVolumeReclaimPolicy: Retain
 storageClassName: local-storage
 local:
  path: /home/vagrant/storage
 nodeAffinity:
  required:
   nodeSelectorTerms:
   - matchExpressions:
    - key: kubernetes.io/hostname
      operator: In
      values:
      - node1

I saved the above configuration with name **jhooq-pv.yml** but you can assign any name of your choice.

In step 1 we have created Persistent Volume for Local Storage, now you need to apply the configuration.

Step 3 - Apply the configuration

BASH

```
kubectl apply -f jhooq-pv.yml
```

BASH

```
persistentvolume/jhooq-demo-pv created
```

Step 4 - Lets check the status of PV

```BASH
kubectl get pv
```

```BASH
NAME           CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM   STORAGECLASS    REASON
jhooq-demo-pv  1Gi        RWO            Retain           Available           local-storage
```

Here is the screen of the status

Great now you have created PV, let's move ahead and create a Persistent Volume claim.

What is Persistent Volume claim?

Persistent volume provides you an abstraction between the consumption of the storage and implementation of the storage.

In the nutshell you can say its a request for storage on behalf of an application which is running on cluster.

How to use Persistent Volume claim(PVC) ?

If you as an application developer wants to use/access Persistent Volume(PV) then you must create a request for storage and it can be done by creating PVC objects.

Step 1 - Alright lets create your first Persistent Volume Claim(jhooq-pvc.yml) -

YAML

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: jhooq-pvc
spec:
  volumeName: jhooq-demo-pv
  storageClassName: local-storage
  volumeMode: Filesystem
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

Save the above configuration with file name of your choice, in mycase I am saving this file with the name **jhooq-pvc.yml**

apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: jhooq-pvc
spec:
 volumeName: jhooq-demo-pv
 storageClassName: local-storage
 volumeMode: Filesystem
 accessModes:
  - ReadWriteOnce
 resources:
  requests:
   storage: 1Gi

Step 2 - Now apply this configuration using the following command

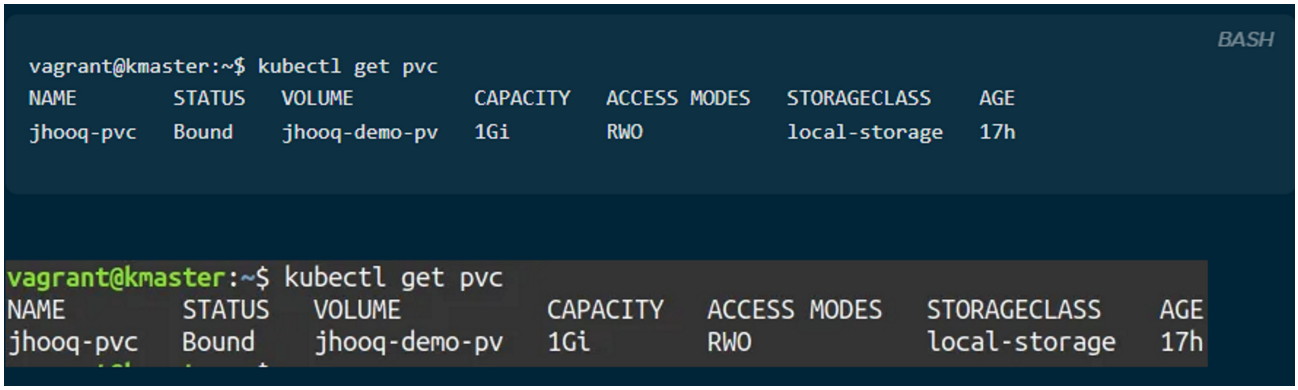
```bash
kubectl create -f jhooq-pvc.yml
```

```bash
persistentvolumeclaim/jhooq-pvc created
```

Step 3 - Lets check our Persistent Volume and persistent Volume Claim

```bash
kubectl get pvc
```

```bash
vagrant@kmaster:~$ kubectl get pvc
NAME        STATUS   VOLUME          CAPACITY   ACCESS MODES   STORAGECLASS    AGE
jhooq-pvc   Bound    jhooq-demo-pv   1Gi        RWO            local-storage   17h
```

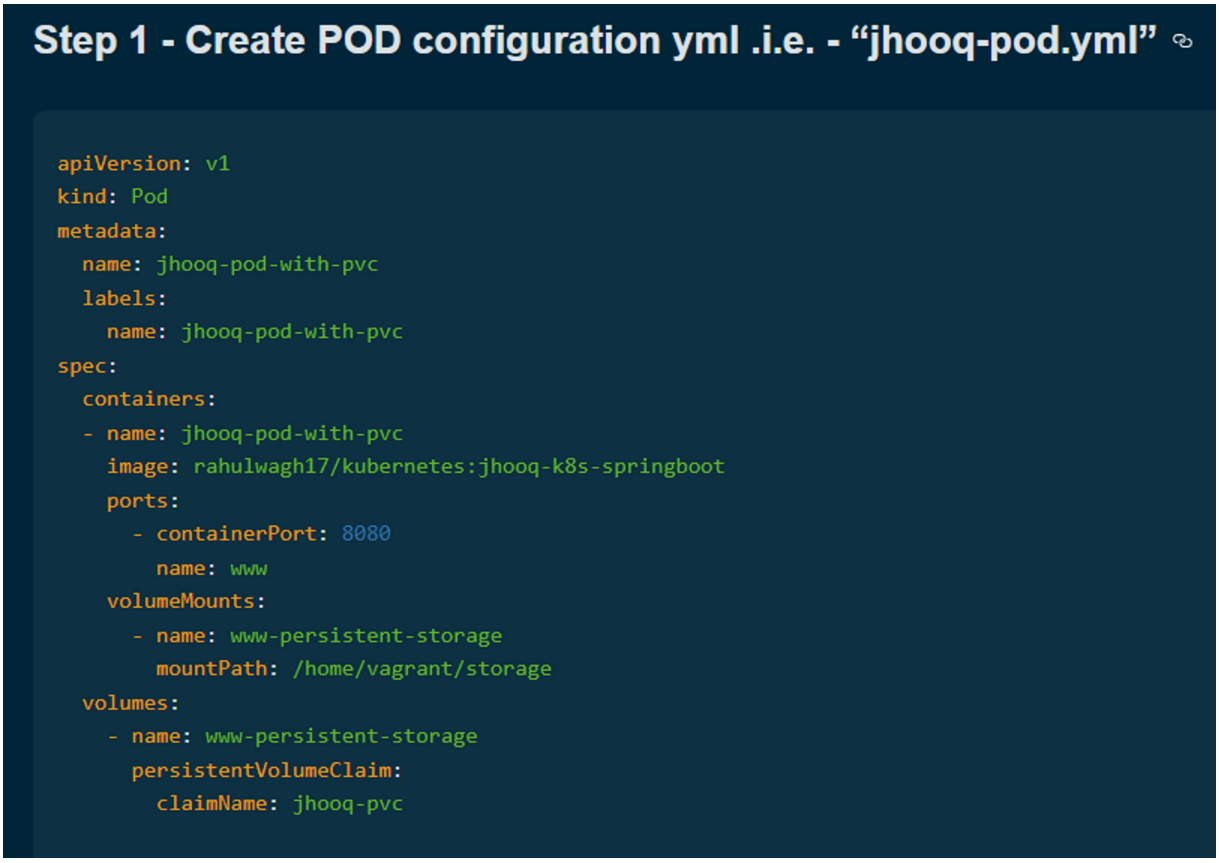


Create a POD using Persistent Volume claim

Now in this step we are going to create a POD using the PV and PVC from the previous steps.

We are going to deploy Spring Boot Docker image but you can use any docker application of your choice. But if you do not have any docker image with you then you can refer to - How to deploy spring boot application in Kubernetes cluster

Step 1 - Create POD configuration yml .i.e. - “jhooq-pod.yml”



```yaml
apiVersion: v1
kind: Pod
metadata:
  name: jhooq-pod-with-pvc
  labels:
    name: jhooq-pod-with-pvc
spec:
  containers:
  - name: jhooq-pod-with-pvc
    image: rahulwagh17/kubernetes:jhooq-k8s-springboot
    ports:
      - containerPort: 8080
        name: www
    volumeMounts:
      - name: www-persistent-storage
        mountPath: /home/vagrant/storage
  volumes:
    - name: www-persistent-storage
      persistentVolumeClaim:
        claimName: jhooq-pvc
```


```yaml
apiVersion: v1
kind: Pod
metadata:
  name: jhooq-pod-with-pvc
  labels:
    name: jhooq-pod-with-pvc
spec:
  containers:
  - name: jhooq-pod-with-pvc
    image: rahulwagh17/kubernetes:jhooq-k8s-springboot
    ports:
      - containerPort: 8080
        name: www
    volumeMounts:
      - name: www-persistent-storage
        mountPath: /home/vagrant/storage
  volumes:
    - name: www-persistent-storage
      persistentVolumeClaim:
        claimName: jhooq-pvc
```

Step 2 : Lets apply the pod configuration

```bash
kubectl apply -f jhooq-pod.yml
```

```bash
pod/jhooq-pod-with-pvc created
```

```
vagrant@kmaster:~/jhooq-demo-pv$ kubectl apply -f jhooq-pod.yml
pod/jhooq-pod-with-pvc created
```

Lets check the pod status

```bash
$kubectl get pod
```

```bash
NAME                READY   STATUS    RESTARTS   AGE
jhooq-pod-with-pvc   1/1     Running   0          8m37s
```

Now you have deployed your pod successfully using the Persistent Volume and Persistent Volume Claim.

Step 3 : Test the microservice deployed under the POD

In this testing step we need to access the microservice which we deployed inside the POD.

To test the POD first we need to find the IP address on which it is running.

Use the following kubectl command to find the IP address of the POD

```bash
$ kubectl get pod -o wide
```

It should return you with (You may get different IP address) -

```
vagrant@node1:~$ kubectl get pod -o wide
NAME                READY   STATUS    RESTARTS   AGE   IP            NODE    NOMINATED NODE   READINESS GATES
jhooq-pod-with-pvc   1/1     Running   0          96m   10.233.90.3   node1   <none>           <none>
```

Ref : https://jhooq.com/