Volumes in Kubernetes

Before we head into persistent volumes let us start with volumes in Kubernetes. Let us look at volumes in Docker first.

Docker containers are meant to be transient in nature, which means they are meant to last only for a short period of time. They're called upon when required to process data and destroyed once finished. The same is true for the data within the container. The data is destroyed, along with the container.

To persist data processed by the containers, we attach a volume to the containers when they are created. The data processed by the container is now placed in this volume, thereby retaining it permanently. Even if the container is deleted, the data generated or processed by it remains.

So how does that work in the Kubernetes world? Just as in Docker, the pods created in Kubernetes are transient in nature. When a pod is created to process data, and then deleted, the data processed by it, gets deleted as well. For this, we attach a volume to the pod. The data generated by the pod is now stored in the volume, and even after the pod is deleted, the data remains.

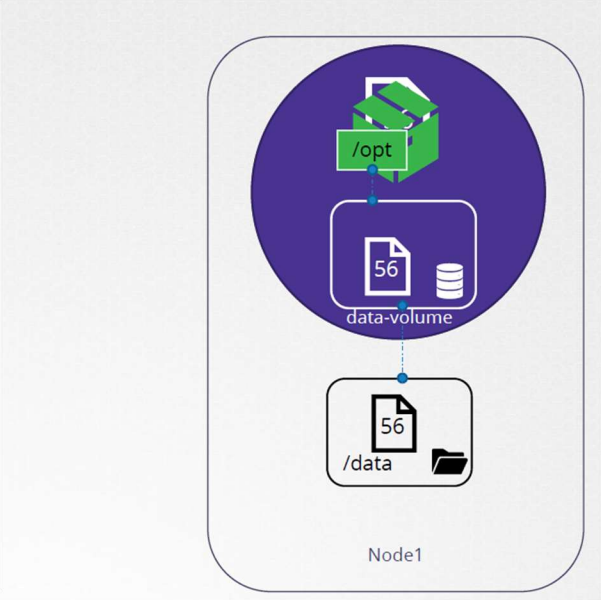
Let's look at a simple implementation of volumes. We have a single node Kubernetes cluster. We create a simple pod that generates a random number between one and hundred, and writes that with file at **/opt/number.out** It then gets deleted along with the random number.

To retain the number generated by the pod, we create a volume. And a volume needs a storage. When you create a volume, you can choose to configure its storage in different ways. We will look at the various options in a bit, but for now we will simply configure it to use a directory on the host. In this case, I specify a path, **/data**, on the host. This way, any files created in the volume would be stored in the directory **data** on my node.

Once the volume is created, to access it from a container we mount the volume to a directory inside the container. We use the volume mounts field in each container to mount the data volume to the directory, **/opt** within the container.

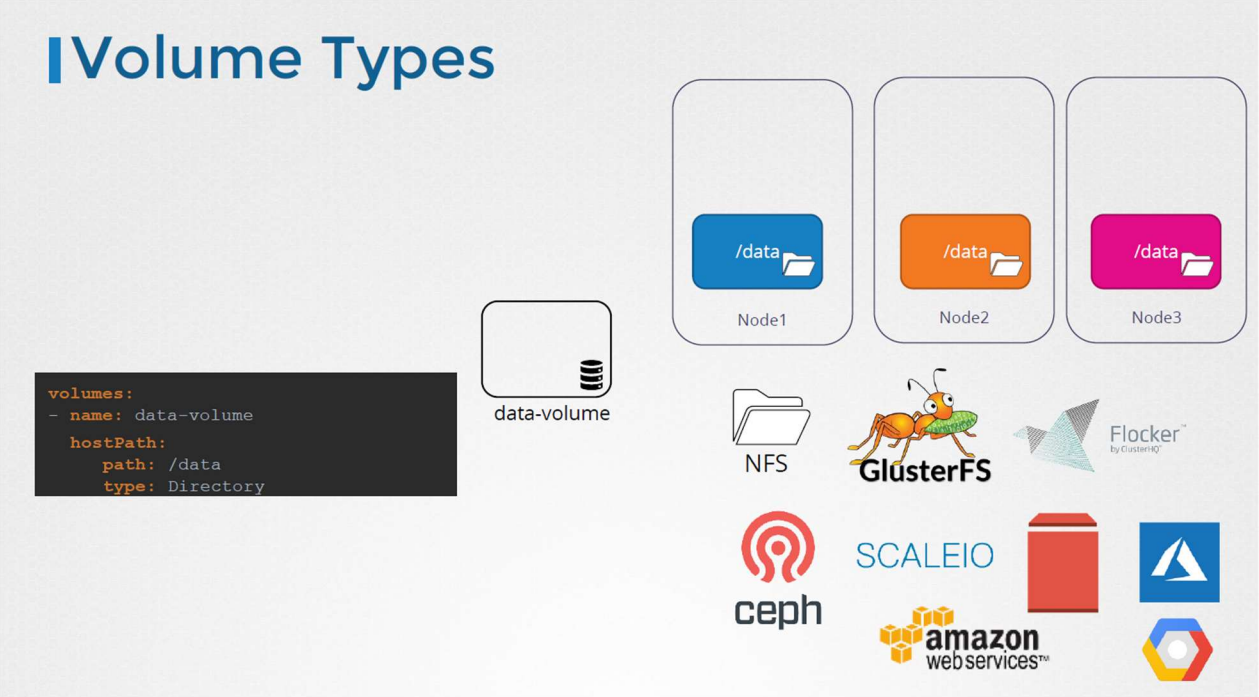
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: random-number-generator
spec:
  containers:
  - image: alpine
    name: alpine
    command: ["/bin/sh","-c"]
    args: ["shuf -i 0-100 -n 1 >> /opt/number.out;"]
    volumeMounts:
    - mountPath: /opt
      name: data-volume

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



The random number will now be written to /opt mount inside the container, which happens to be on the data volume, which is in fact the data directory on the host. When the pod gets deleted, the file with the random number still lives on the host.

Let's take a step back and look at the volume storage options. We just used the host path option to configure it directly on the host as storage space for the volume. Now that works fine on a single node, however, it is not recommended for use in a multi node cluster.This is because the pods would use the /data directory on all the nodes, and expect all of them to be the same and have the same data. Since they're on different servers, they're in fact, not the same. Unless you configure some kind of external replicated cluster storage solution. Kubernetes supports several types of different storage solutions, such as NFS, cluster affairs, Flocker, fiber channel, Ceph FS, scale io, or public cloud solutions like AWS, EBS, Azure desk, or file, or Google's Persistent Desk.



For example, to configure an AWS elastic block store volume as the storage option for the volume, we replaced host path field of the volume with the AWS Elastic block store field, along with the volume ID and file system type. The volume storage will now be on AWS EBS.

```
volumes:
- name: data-volume
  awsElasticBlockStore:
    volumeID: <volume-id>
    fsType: ext4
```

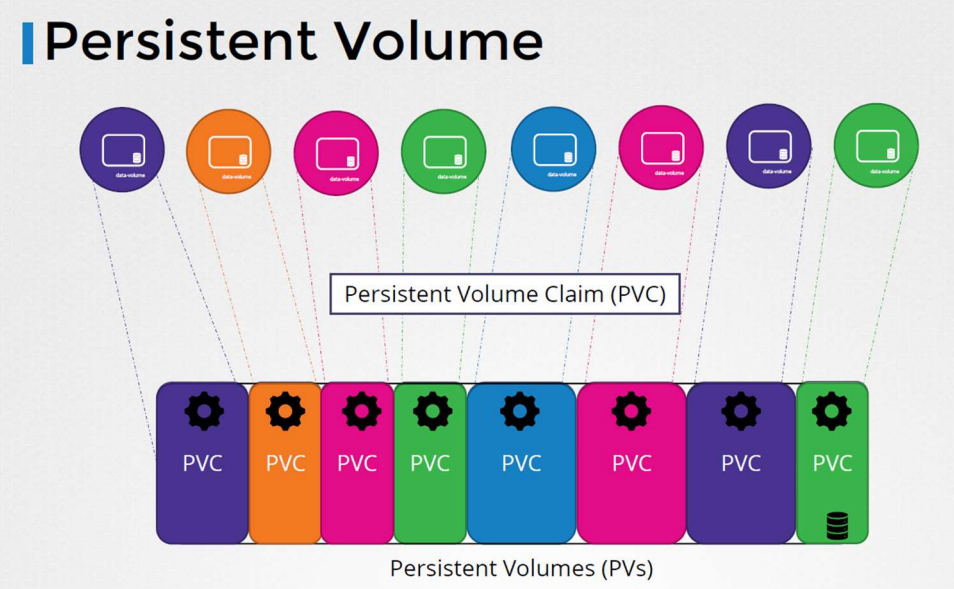
Persistent volumes

In the last lecture, we learned about volumes. Now we will discuss persistent volumes in Kubernetes. When we created volumes in the previous section, we configured volumes within the pod definition file, so every configuration information required to configure storage for the volume goes within the pod definition file.

Now, when you have a large environment with a lot of users deploying a lot of pods, the users would have to configure storage every time for each pod. Whatever storage solution is used, the users who deploys the pods would have to configure that on all pod definition files in his environment.

Every time there are changes to be made the user would have to make them on all of his pods. Instead, you would like to manage storage more centrally. You would like it to be configured in a way that an administrator can create a large pool of storage and then have users carve out pieces from it as required. That is where persistent volumes can help us.

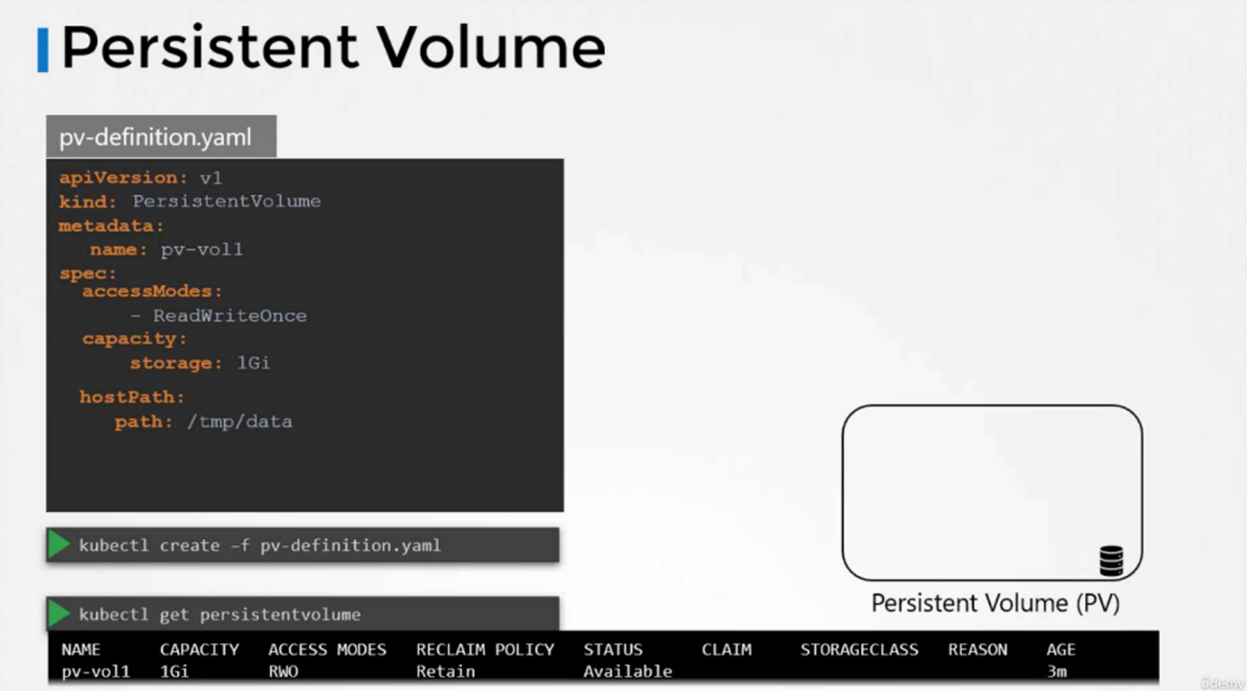
A persistent volume is a cluster-wide pool of storage volumes configured by an administrator to be used by users deploying applications on the cluster. The users can now select storage from this pool using persistent volume claims.



Let us now create a persistent volume. We start with the base template and update the API version. Set the kind to **PersistentVolume** and name it PV-Vol 1. Under the spec section, specify the access modes. Access mode defines how a volume should be mounted on the hosts, whether in a read-only mode or read/write mode, et cetera. The supported values are ReadOnlyMany, ReadWriteOnce, or ReadWriteMany mode.

Next is the capacity. Specify the amount of storage to be reserved for this persistent volume, which is set to 1 GB here.

Next comes the volume type. We will start with the hostPath option that uses storage from the node's local directory. Remember, this option is not to be used in a production environment. To create the volume, run Kube Control, create command, and to list the created volume, run the Kube Control, get persistent volume command.



Replace the host path option with one of the supported storage solutions as we saw in the previous lecture, like AWS Elastic Block Store, et cetera.