Step 1 - Deploy NFS Server
NFS is a protocol that allows nodes to read/write data over a network. The protocol works by having a master node running the NFS daemon and stores the data. This master node makes certain directories available over the network.

Clients access the masters shared via drive mounts. From the viewpoint of applications, they are writing to the local disk. Under the covers, the NFS protocol writes it to the master.

An NFS (Network File System) server is a system that shares its file system with other systems on the network using the NFS protocol. NFS is a distributed file system protocol that allows files and directories to be shared across a network, as if they were stored on a local file system.

When an NFS server is set up, it exports one or more directories to the network, making them available to other systems that have been granted access to those directories. NFS clients can then mount these shared directories, giving them access to the files and directories contained within. This allows users and applications to access and manipulate shared files as if they were stored locally, without the need for manual copying or syncing of data.

Setting up an NFS server typically involves installing and configuring NFS software on the server system, configuring the exported directories, and setting up appropriate access control to restrict or allow access to those directories by specific hosts or networks. On the client side, NFS software is typically already installed on most Linux and Unix-like systems, allowing users to mount and access shared NFS directories with the mount command or by configuring the system to automatically mount them at boot time.

Overall, NFS provides a convenient and efficient way to share and access files across a network, making it a popular choice for many organizations and users.

Task
In this scenario, and for demonstration and learning purposes, the role of the NFS Server is handled by a customised container. The container makes directories available via NFS and stores the data inside the container. In production, it is recommended to configure a dedicated NFS Server.

Start the NFS using the command

The NFS server exposes two directories, data-0001 and data-0002. In the next steps, this is used to store data.

This command starts a new Docker container running the katacoda/contained-nfs-server image as an NFS server.

Here is a breakdown of the different options used in the docker run command:

-d: This option tells Docker to run the container in the background, detached from the terminal.

--net=host: This option specifies that the container should share the host's network namespace, allowing it to use the host's IP address and network interfaces directly.

--privileged: This option gives the container full access to the host system's resources, allowing it to perform actions that would normally be restricted to the host system itself.

--name nfs-server: This option assigns a name to the container, in this case "nfs-server".

katacoda/contained-nfs-server:centos7: This is the name of the Docker image to use for the container, in this case the katacoda/contained-nfs-server image based on CentOS 7.

/exports/data-0001 /exports/data-0002: These are the directories that the NFS server will export and make available to other systems on the network. In this case, the directories /exports/data-0001 and /exports/data-0002 are being exported.
Overall, this command starts a new container running an NFS server with the two directories /exports/data-0001 and /exports/data-0002 exported and available for use by other systems on the network. The --privileged and --net=host options are used to give the container full access to the host system's resources and network, which is necessary for the NFS server to function correctly.

Step 2 - Deploy Persistent Volume

In Kubernetes, a Persistent Volume (PV) is a cluster-wide storage resource that can be dynamically provisioned and bound to a Persistent Volume Claim (PVC) by a pod or container. A Persistent Volume provides a way to store data outside the pod or container that uses it, allowing the data to persist even if the pod or container is deleted or recreated.

A Persistent Volume is created by a cluster administrator and represents a piece of storage in the cluster. It is defined by its capacity, access mode, and storage class. The capacity specifies the amount of storage available for use by the pod or container. The access mode specifies the type of access that is allowed to the storage resource, such as read-write or read-only. The storage class specifies the type of storage to be used, such as local storage, network-attached storage (NAS), or cloud storage.

A Persistent Volume Claim (PVC) is a request for storage made by a pod or container. A PVC specifies the amount of storage required, the access mode, and the storage class. When a PVC is created, Kubernetes will automatically search for a suitable available Persistent Volume that matches the PVC's requirements and bind the PVC to the selected Persistent Volume.

Once a PVC is bound to a Persistent Volume, the pod or container can use the Persistent Volume for storage. If the pod or container is deleted or recreated, the Persistent Volume will persist the data stored within it and can be reused by the new pod or container that is created. This ensures that data is preserved across pod and container restarts, and enables stateful applications to run reliably in Kubernetes.

Deploying a persistent volume in Kubernetes is a crucial step in ensuring that data can be stored and accessed by the various pods and containers that make up a Kubernetes application.

Here are the basic steps involved in deploying a persistent volume in Kubernetes:

Create a Persistent Volume (PV) definition file that describes the characteristics of the volume you want to create, such as the storage capacity, access mode, and storage class. The PV definition file is typically written in YAML format and can be created using a text editor or generated by a tool like kubectl.

Apply the PV definition file using kubectl apply to create the actual persistent volume in the Kubernetes cluster.

Create a Persistent Volume Claim (PVC) definition file that specifies the storage requirements of the pod or container that will use the persistent volume. The PVC definition file is also typically written in YAML format and can be created using a text editor or generated by a tool like kubectl.

Apply the PVC definition file using kubectl apply to request a claim to the persistent volume. Kubernetes will automatically bind the PVC to an available PV that matches the requested storage requirements.

Create a pod or container definition file that specifies the use of the PVC for storage. The pod or container definition file is also typically written in YAML format and can be created using a text editor or generated by a tool like kubectl.

Apply the pod or container definition file using kubectl apply to create the pod or container that will use the persistent volume for storage.

By following these steps, you can deploy a persistent volume in Kubernetes and use it to store and access data across multiple pods and containers in your application.

Step 2 - Deploy Persistent Volume
For Kubernetes to understand the available NFS shares, it requires a PersistentVolume configuration. The PersistentVolume supports different protocols for storing data, such as AWS EBS volumes, GCE storage, OpenStack Cinder, Glusterfs and NFS. The configuration provides an abstraction between storage and API allowing for a consistent experience.
In the case of NFS, one PersistentVolume relates to one NFS directory. When a container has finished with the volume, the data can either be Retained for future use or the volume can be Recycled meaning all the data is deleted. The policy is defined by the persistentVolumeReclaimPolicy option.

In Kubernetes, persistentVolumeReclaimPolicy is a property of a Persistent Volume that determines what happens to the volume's data when the PV is released or deleted.

There are three possible values for persistentVolumeReclaimPolicy:

Retain: This policy preserves the data in the Persistent Volume after the PVC that uses it is deleted. If a new PVC is created that matches the specifications of the deleted PVC, the Persistent Volume will be reused.

Delete: This policy deletes the data in the Persistent Volume when the PVC that uses it is deleted. The PV will be released and made available for use by other PVCs, but the data stored in it will be permanently deleted.

Recycle: This policy deletes the contents of the Persistent Volume when the PVC that uses it is deleted, but does not delete the Persistent Volume itself. The PV is then reformatted and made available for use by other PVCs.

The persistentVolumeReclaimPolicy property can be set in the Persistent Volume's specification when it is created or updated. The default value for persistentVolumeReclaimPolicy is Delete.

It is important to choose the appropriate persistentVolumeReclaimPolicy for your application's needs. For example, if your application requires data persistence even after the PVC is deleted, you should choose the Retain policy. If your application does not require data persistence, or if the data can be recreated easily, you can choose the Delete policy.

For structure is:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: <friendly-name>
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
    - ReadWriteMany
  persistentVolumeReclaimPolicy: Recycle
  nfs:
    server: <server-name>
    path: <shared-path>
```

The spec defines additional metadata about the persistent volume, including how much space is available and if it has read/write access.

This is a YAML manifest file for creating a Kubernetes Persistent Volume (PV) object that uses NFS storage. Here's what each section of the manifest does:

apiVersion and kind: These specify the API version and object type for the Kubernetes object that will be created. In this case, the object type is PersistentVolume, which defines a cluster-wide storage resource.

metadata: This section defines metadata for the Persistent Volume, such as a friendly name for the resource.

spec: This section defines the specifications for the Persistent Volume. It includes:

capacity: This specifies the capacity of the Persistent Volume, which in this case is 1 GiB of storage.

accessModes: This specifies the access modes for the Persistent Volume, which can be either ReadWriteOnce or ReadWriteMany. ReadWriteOnce means that the Persistent Volume can be mounted as read-write by a single node at a time, while ReadWriteMany means that the Persistent Volume can be mounted as read-write by multiple nodes simultaneously.

persistentVolumeReclaimPolicy: This specifies the reclaim policy for the Persistent Volume, which in this case is Recycle. This means that the contents of the Persistent Volume will be deleted when the corresponding Persistent Volume Claim (PVC) is deleted, and the Persistent Volume will be reformatted and made available for reuse.

nfs: This specifies that the Persistent Volume will use NFS storage. It includes:

server: This specifies the hostname or IP address of the NFS server.

path: This specifies the shared path on the NFS server where the data will be stored.

Overall, this manifest file creates a Kubernetes Persistent Volume object that can be used by pods or containers to store data on an NFS server. The persistentVolumeReclaimPolicy is set to Recycle, which means that the data stored in the volume will be deleted when the corresponding PVC is deleted, but the volume itself will be reformatted and made available for reuse.

Task
Create two new PersistentVolume definitions to point at the two available NFS shares.
Once created, view all PersistentVolumes in the cluster using kubectl get pv

```
controlplane $ cat nfs-0001.yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: nfs-0001
spec:
  capacity:
    storage: 2Gi
  accessModes:
    - ReadWriteOnce
    - ReadWriteMany
  persistentVolumeReclaimPolicy: Recycle
  nfs:
    server: 10.0.0.9
    path: /exports/data-0001
controlplane $ kubectl create -f nfs-0001.yaml
persistentvolume/nfs-0001 created
controlplane $ cat nfs-0002.yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: nfs-0002
spec:
  capacity:
    storage: 5Gi
  accessModes:
    - ReadWriteOnce
    - ReadWriteMany
  persistentVolumeReclaimPolicy: Recycle
  nfs:
    server: 10.0.0.9
    path: /exports/data-0002
controlplane $ kubectl create -f nfs-0002.yaml
persistentvolume/nfs-0002 created
controlplane $ kubectl get pv
NAME       CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM   STORAGECLASS   REASON   AGE
nfs-0001   2Gi        RWO,RWX        Recycle          Available                                   27s
nfs-0002   5Gi        RWO,RWX        Recycle          Available                                   15s
controlplane $
```

The kubectl create -f command is used to create Kubernetes objects from a YAML or JSON file.
When you run kubectl create -f nfs-0002.yaml, Kubernetes will read the YAML manifest from the file and create the corresponding Persistent Volume object. The nfs-0002 name in the YAML file likely refers to a specific NFS volume that is being mounted as a Persistent Volume in Kubernetes.

After running the kubectl create -f command, you can use the kubectl get pv command to verify that the Persistent Volume has been created.

Step 4 - Use Volume
When a deployment is defined, it can assign itself to a previous claim. The following snippet defines a volume mount for the directory /var/lib/mysql/data which is mapped to the storage mysql-persistent-storage. The storage called mysql-persistent-storage is mapped to the claim called claim-mysql.

```
spec:
   volumeMounts:
     - name: mysql-persistent-storage
       mountPath: /var/lib/mysql/data
volumes:
  - name: mysql-persistent-storage
    persistentVolumeClaim:
      claimName: claim-mysql
```

In this YAML snippet, the volumeMounts field specifies how the volumes should be mounted inside the container. In this case, a volume called mysql-persistent-storage is mounted to the path /var/lib/mysql/data. This means that any data written to this directory inside the container will be persisted to the Persistent Volume associated with the mysql-persistent-storage volume.

The volumes field defines the list of volumes that the container will have access to. In this case, a volume named mysql-persistent-storage is defined, and its source is specified as a Persistent Volume Claim named claim-mysql using the persistentVolumeClaim field. This means that Kubernetes will dynamically provision a Persistent Volume and map it to the mysql-persistent-storage volume based on the specifications in the Persistent Volume Claim claim-mysql.

This ensures that the container has access to a Persistent Volume for storing data, which can be dynamically provisioned or reused across deployments as needed. By using a Persistent Volume Claim to request a Persistent Volume, you can separate the concerns of storage provisioning from application deployment, making it easier to manage and scale your applications.

Task
Launch two new Pods with Persistent Volume Claims. Volumes are mapped to the correct directory when the Pods start allowing applications to read/write as if it was a local directory.

```
controlplane $ kubectl create -f pod-mysql.yaml
pod/mysql created
controlplane $ kubectl create -f pod-www.yaml
pod/www created
```

```
controlplane $ cat pod-mysql.yaml
apiVersion: v1
kind: Pod
metadata:
  name: mysql
  labels:
    name: mysql
spec:
  containers:
  - name: mysql
    image: openshift/mysql-55-centos7
    env:
      - name: MYSQL_ROOT_PASSWORD
        value: yourpassword
      - name: MYSQL_USER
        value: wp_user
      - name: MYSQL_PASSWORD
        value: wp_pass
      - name: MYSQL_DATABASE
        value: wp_db
    ports:
      - containerPort: 3306
        name: mysql
    volumeMounts:
      - name: mysql-persistent-storage
        mountPath: /var/lib/mysql/data
  volumes:
    - name: mysql-persistent-storage
      persistentVolumeClaim:
        claimName: claim-mysql
```

In this example, a Kubernetes pod object is defined for running a MySQL container. The pod specifies a volume mount for the directory /var/lib/mysql/data, which is mapped to the Persistent Volume Claim called claim-mysql. The pod also specifies a volume object with the name mysql-persistent-storage, which is linked to the Persistent Volume Claim using the persistentVolumeClaim field.

When the pod is created, Kubernetes will dynamically provision a Persistent Volume for the specified Persistent Volume Claim if one does not already exist. If a matching Persistent Volume already exists, it will be used instead. Once the Persistent Volume is provisioned, it will be mounted to the specified directory in the container, allowing data to persist even if the container is terminated or restarted.

```
controlplane $ cat pod-www.yaml
apiVersion: v1
kind: Pod
metadata:
  name: www
  labels:
    name: www
spec:
  containers:
  - name: www
    image: nginx:alpine
    ports:
      - containerPort: 80
        name: www
    volumeMounts:
      - name: www-persistent-storage
        mountPath: /usr/share/nginx/html
  volumes:
    - name: www-persistent-storage
      persistentVolumeClaim:
        claimName: claim-http
```

In this example, a Kubernetes pod object is defined for running an Nginx container with a volume mount for the directory /usr/share/nginx/html. The volume mount is linked to the Persistent Volume Claim called claim-http using the volume object with the name www-persistent-storage.

When the pod is created, Kubernetes will dynamically provision a Persistent Volume for the specified Persistent Volume Claim if one does not already exist. If a matching Persistent Volume already exists, it will be used instead. Once the Persistent Volume is provisioned, it will be mounted to the specified directory in the container, allowing data to persist even if the container is terminated or restarted. In this case, the Persistent Volume Claim is used for storing the HTML content of the Nginx web server.

You can see the status of the Pods starting using kubectl get pods

If a Persistent Volume Claim is not assigned to a Persistent Volume, then the Pod will be in Pending mode until it becomes available. In the next step, we'll read/write data to the volume.

```
controlplane $ kubectl get pods
NAME    READY   STATUS    RESTARTS   AGE
mysql   0/1     Pending   0          8m52s
www     0/1     Pending   0          8m40s
```