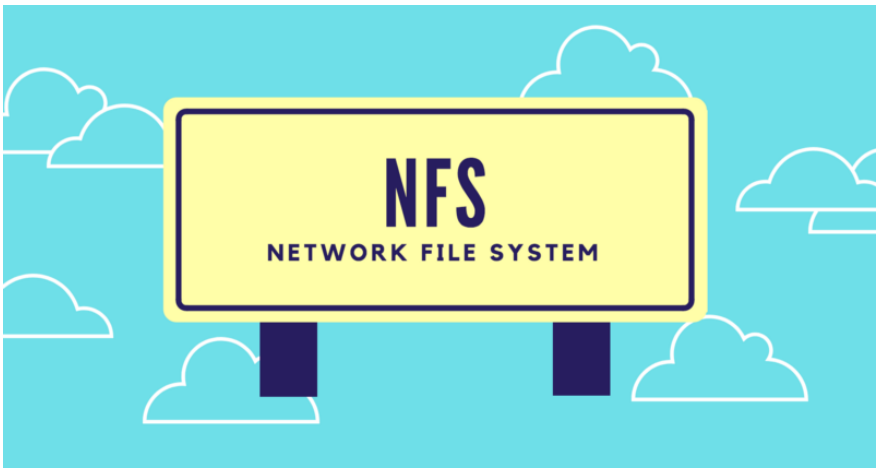
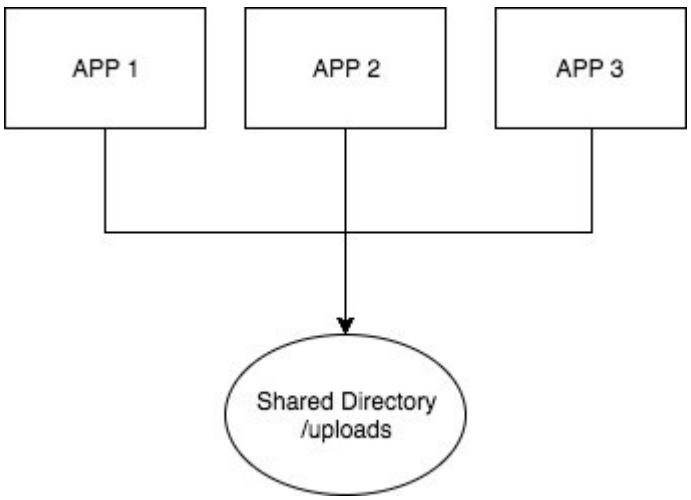
Jul 30 · 5 min read

NFS Persistent Volumes with Kubernetes — A Case Study



Recently, The Platformer team was tasked with migrating an entire infrastructure of one of our clients to the Google Cloud. One of the requirements they requested from us was to containerize their applications and deploy them in a cluster in Google Kubernetes Engine. Here's the architecture of their application.

Architecture



As shown in the diagram, they have 3 applications, each running separately in a VM and have a shared directory called /uploads which allows all 3 of the applications to read and write data on the said location. With containerization and kubernetes migration, they expect to run multiple replicas of each application and all of them should have read write access to a shared volume. This blog post discusses the solution to this problem.

We use DeploymentSets instead of StatefulSets for this particular use case as the pods don't require to store their state uniquely in this architecture. They just require a shared volume to store common data.

NFS Server with Google Persistent Disks

Kubernetes Volumes have 3 access types.

ReadWriteOnce (RWO), R*eadOnlyOnce(ROO)* and R*eadWriteMany(RWX)*. As the names suggest, the first access type only allows a single node to do read writes, second one allows multiple nodes to read only and the third allows read writes on multiple nodes. So the solution is pretty easy right? Create a Persistent Volume with ReadWriteMany access type.

Well, It's not straight forward as it sounds. If you refer the official Google Documentation here, you will notice that in ReadWriteMany access type, **PersistentVolumes that are backed by Compute Engine persistent disks are not supported.**

There are alternative solutions for this like Google Filestore, which can be found in their documentation here. However, you should take note that Google Filestore is developed targeting large scale file system use cases, as the minimum size of an instance is 1TB which is quite expensive for a small scale company with very little data trying to meet the requirement addressed in this post.

So the ideal solution for this problem is to create a Network File System(NFS) Server with Google Compute Engine persistent disks and mount them to your containers.

So let's get started.

Create Persistent Disk in Google Compute Engine

First you need to create a persistent disk in your Google Cloud Console. Open the web console, or your terminal authenticated with gcloud and run the following command.

```
gcloud compute disks create --size=10GB --zone=us-east1-b
gce-nfs-disk
```

Alternatively, you can create this manually using the GCP UI as well. Make sure to change the zone depending on the zone of your Kubernetes is cluster is in.

Create NFS Server in GKE

With the disk created, let's create the NFS server. This is pretty straightforward and you can apply this configuration straightaway. Create a file called 001-nfs-server.yaml and paste the following code.

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: nfs-server
spec:
  replicas: 1
  selector:
    matchLabels:
      role: nfs-server
  template:
    metadata:
      labels:
        role: nfs-server
    spec:
      containers:
      - name: nfs-server
        image: gcr.io/google_containers/volume-nfs:0.
        ports:
          - name: nfs
            containerPort: 2049
          - name: mountd
            containerPort: 20048
```

001-nfs-server.yaml

Make sure under *gcePersistentDisk → pdName*, you give the name of the persistent disk you created in Google Compute Engine.

You can apply this configuration by running the following.

```
kubectl apply -f 001-nfs-server.yaml
```

Create NFS Service

This is pretty straight forward as well. Just copy this to a file called 002-nfs-server-service.yaml.

```yaml
apiVersion: v1
kind: Service
metadata:
  name: nfs-server
spec:
  # clusterIP: 10.3.240.20
  ports:
    - name: nfs
      port: 2049
    - name: mountd
```

002-nfs-server-service.yaml

Let's apply this configuration as well.

```
kubectl apply -f 002-nfs-server-service.yaml
```

After creating the service, you need to get the clusterIP of the service. To do that run the following command and you will get a result as shown below.

```
kubectl get svc nfs-server
```

```
NAME         TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)                    AGE
nfs-server   ClusterIP   10.23.241.98   <none>        2049/TCP,20048/TCP,111/TCP   3h
```

nfs-server-service information

Note down the clusterIP of this as it will be required for the next part.

Create Persistent Volume and Persistent Volume Claims

Okay, now we create a persistent volume and a persistent volume claim in kubernetes. Persistant Volume (PV) is a piece of storage in the cluster that has been provisioned by an administrator. It is a resource in the cluster where as a Persistent Volume Claim (PVC) is a request for storage by a user. It is similar to a pod. Just like how pods consume

node resources, PVCs consume PV resources and they can request how much storage they need from the PV. You can check the official documentation here for more information on this.

Note that you have to paste the ClusterIP you copied in the previous section here.

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: nfs
spec:
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteMany
  nfs:
    server: <ClusterIP>
    path: "/"

---
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
```

003-pv-pvc.yaml

You can create a file called *003-pv-pvc.yaml* and run the following command to create your volume and volume claim.

```
kubect apply -f 003-pv-pvc.yaml
```

Notice here that storage is 10Gi on both PV and PVC. It's 10Gi on PV because the Compute Engine persistent disk we created was of the size 10Gi. It is linked to the PV as a Network File System server using the *nfs-server* deployment we made in the earlier section.

You can have any storage value for PVC as long as you don't exceed the storage value defined in PV.

That's pretty much it. You are now ready to use the shared volume between your pods.

Bonus: Configuring Pods with PVC

You can check the following busy box example on how to bind a PVC to a deployment. You create a volume in deployment spec and add that volume to volumeMount in container spec and you are good to go!

Additionally, there's a Helm Chart available for NFS Server with Kuberentes which can be found out on the following link. Helm Charts is a topic for another day and if you don't understand what it is, just ignore this paragraph.

helm/charts

charts - Curated applications for Kubernetes

github.com

I hope this helped you. Clap for more.
Cheers!