GKE Storage

Author: Heng Wei
Date: 2020/05/23
Version: 1.0

Kubernetes有多种存储的方式，本文将介绍在GKE上常使用的:

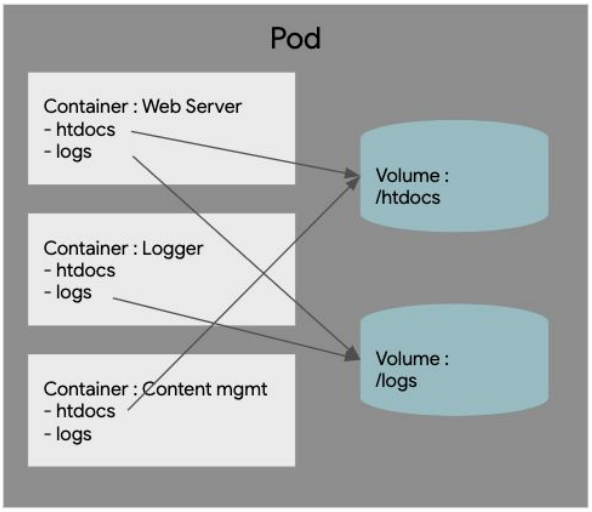
1. Volumes
   a. GCE-PD
   b. NFS
2. Persistent Volumes
   a. GCE PD
   b. NFS
3. GCE-PD CSI Driver

1 Volume

大家知道，在容器中文件是临时存放在磁盘上的。当容器删除，容器对应在磁盘上的临时文件会被删除，这会对一些应用的运行造成问题。 比如：

1. 当容器崩溃时，kubenetes会重启这个容器，但这个容器中的文件会丢失，因为崩溃的容器将被删除，创建一个新的容器。
2. 当在一个 Pod 中运行多个容器时，需要在这些容器间共享文件。 Kubernetes通过Volume来解决这些问题。

Volume在Pod中的应用如下图：



Kubernetes支持很多种Volume，GKE支持GCP相关以及通用的一些Volume，比如：

Temp	Variable	Local	Network

emptyDir	Secret ConfigMap	hostPath local	iSCSI NFS gcePersistentDisk persistentDisk CSI

本章将介绍和GCP服务相关的gcePersistentDisk和Filestore支持的nfs。后面两章介绍persistentVolumeClaim和CSI Driver。

1.1 GCE Persistent Disk

GKE的Pod可以直接采用GCE Persistent Disk作为volume。具体做法如下：

- 创建GCE Disk：

```
gcloud compute disks create --size=500GB --zone=us-central1-c my-data-disk
```

- 应用到Pod上

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-with-pd
spec:
  containers:
  - image: nginx
    name: pod-with-pd
    volumeMounts:
    - mountPath: /test-pd
      name: test-volume
  volumes:
  - name: test-volume
    gcePersistentDisk:
      pdName: my-data-disk
      fsType: ext4
```

- 查看pod情况

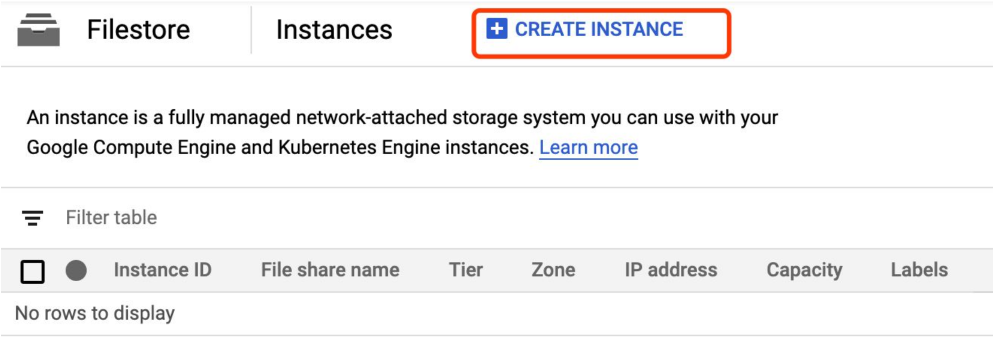
```
$ kubectl describe pod
Name:         pod-with-pd
Namespace:    default
...
Containers:
  pod-with-pd:
...
    Mounts:
      /test-pd from test-volume (rw)
...
Volumes:
  test-volume:
    Type:   GCEPersistentDisk (a Persistent Disk resource in Google Compute Engine)
    PDName:     my-data-disk
    FSType:     ext4
    Partition:  0
    ReadOnly:   false
...
Events:
  Type    Reason                  Age   From  Message

...
  Normal  SuccessfulAttachVolume  61s   attachdetach-controller
AttachVolume.Attach succeeded for volume "test-volume"

...
```

1.2 NFS

1.2.1 创建Filestore-NFS服务

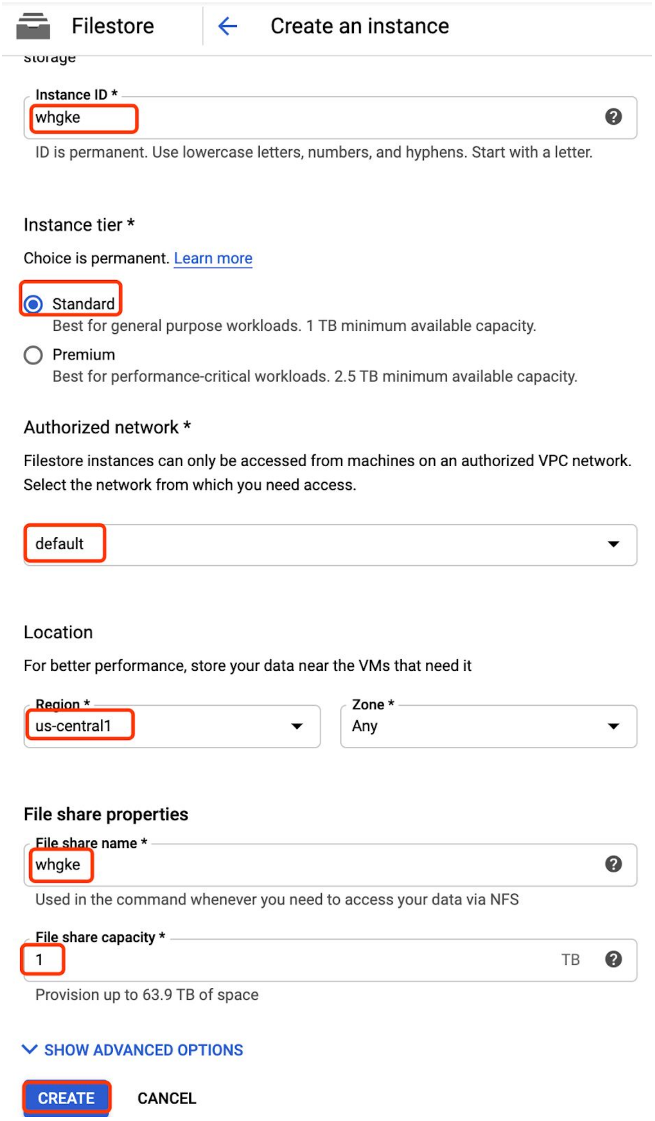
Google Cloud上有NFS的服务：Filestore。

- 创建Filestore：



An instance is a fully managed network-attached storage system you can use with your Google Compute Engine and Kubernetes Engine instances. Learn more

□	●	Instance ID	File share name	Tier	Zone	IP address	Capacity	Labels
No rows to display								



Filestore ← Create an instance

storage

Instance ID *: whgke

ID is permanent. Use lowercase letters, numbers, and hyphens. Start with a letter.

Instance tier *

Choice is permanent. Learn more

● Standard
Best for general purpose workloads. 1 TB minimum available capacity.

○ Premium
Best for performance-critical workloads. 2.5 TB minimum available capacity.

Authorized network *

Filestore instances can only be accessed from machines on an authorized VPC network. Select the network from which you need access.

default

Location

For better performance, store your data near the VMs that need it

Region *: us-central1

Zone *: Any

File share properties

File share name *: whgke

Used in the command whenever you need to access your data via NFS

File share capacity *: 1 TB

Provision up to 63.9 TB of space

1.2.2 Pod直接挂载nfs

- 在pod中挂载创建好的nfs服务

```
apiVersion: v1
kind: Pod
metadata:
  name: test-nfs
spec:
  containers:
  - image: nginx
    name: test-nfs
    volumeMounts:
    - mountPath: /usr/share/nginx/htmlt
      name: test-nfs
  volumes:
  - name: test-nfs
    # nfs
    nfs:
      server: 10.161.227.98
      path: "/whgke"
```

- 查看pod

```
$ kubectl describe pod test-nfs
Name:         test-nfs
Namespace:    default
...
Containers:
  test-nfs:
...
    Mounts:
      /usr/share/nginx/html from test-nfs (rw)
Volumes:
  test-nfs:
    Type:      NFS (an NFS mount that lasts the lifetime of a pod)
    Server:    10.161.227.98
    Path:      /whgke
    ReadOnly:  false
...
```

1.2.3 Deployment直接挂载nfs

通过部署nfs的volume在Deployment中，可以实现deployment中所有的Pod共享相同的文件。

- 部署Deployment，挂载nfs

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: deploy-nfs
  labels:
    app: deploy-nfs
spec:
  selector:
    matchLabels:
      app: deploy-nfs
  replicas: 4
  template:
    metadata:
      labels:
        app: deploy-nfs
    spec:
      containers:
      - image: nginx
        name: test-nfs
        volumeMounts:
        - mountPath: /usr/share/nginx/html
          name: test-nfs
      volumes:
      - name: test-nfs
        nfs:
          server: 10.161.227.98
          path: "/whgke"
```

2 Persistent Volume

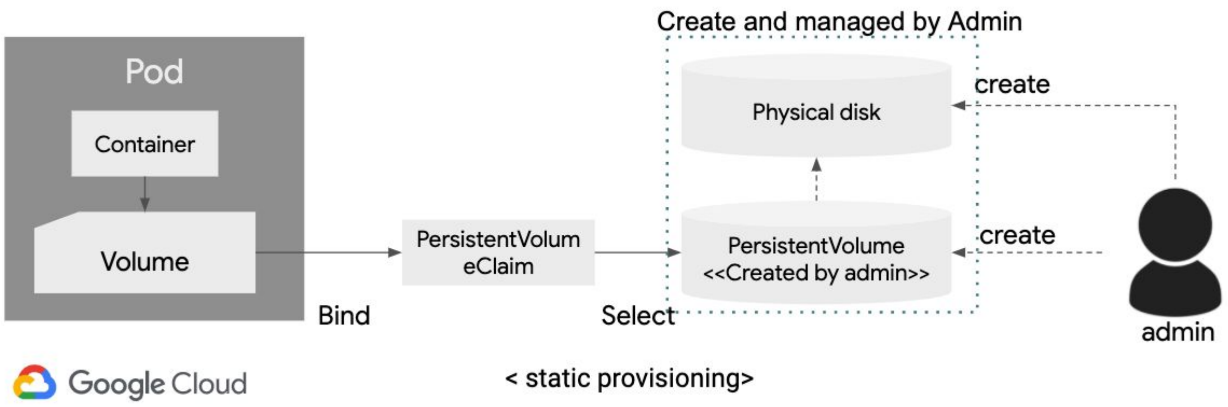
在Volume的方案中，Pod都是静态配置Disk或nfs的信息。这要求用户对底层的存储信息了解的非常清楚。这使得部署非常的不方便。persitentVolume可以屏蔽底层存储的信息，方便用户的部署。本章将介绍GKE环境中如何通过静态、动态的方式部署persistentVolume。

2.1 GCE Persistent Disk

通过gce-pd的privisioner，对storageClass/PV/PVC进行说明，创建静态或动态的PV和PVC，并在Pod里调用，就可以在pod中使用GCE的persistent disk。

2.1.1 静态部署

静态部署的需要静态的创建PV和PVC，从而创建PVC和Disk的映射关系，并在Pod中使用的方法。如下图：



2.1.1.1 Pod应用pvc的静态部署

具体实现方式如下：

- 创建Disk：

```
gcloud compute disks create for-pv-ssd-01 \
  --zone us-central1-c --size 10G --type pd-ssd
```

- 创建storageClass，由于采用的是SSD的disk，这里定义一个fast的storageClass

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: fast
provisioner: kubernetes.io/gce-pd
parameters:
  type: pd-ssd
  zone: us-central1-c
```

- 创建静态的PV，建立PV和Disk的mapping

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-ssd
  labels:
    type: pv-ssd
spec:
  storageClassName: fast
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  gcePersistentDisk:
    pdName: for-pv-ssd-01
    fsType: ext4
```

- 创建静态的PVC

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc1
spec:
  storageClassName: fast
  volumeName: pv-ssd
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10G
  selector:
    matchLabels:
      type: pv-ssd
```

- Pod中调用PVC

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-with-static-pvc
spec:
  containers:
    - name: nginx
      image: nginx
      volumeMounts:
      - mountPath: "/usr/share/nginx/html"
        name: static-pvc
  volumes:
    - name: static-pvc
      persistentVolumeClaim:
        claimName: pvc1
```

- 查看PV/PVC的状态

```
$ kubectl get pv
NAME     CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM
pv-ssd   10Gi       RWO            Retain           Bound    default/pvc1

STORAGECLASS   REASON   AGE
fast                    5m16s

$ kubectl get pvc
NAME   STATUS   VOLUME   CAPACITY   ACCESS MODES   STORAGECLASS   AGE
pvc1   Bound    pv-ssd   10Gi       RWO            fast           4m28s
```

这种方式和前面静态调用gcePersisitentDisk的方式类似，都是静态的mapping。只是通过静态pv和disk的mapping代替了vlome和disk的mapping。

2.1.1.2 Deployment应用ReadOnlyMany Disk静态部署模式

在PV和PVC中有三种accessMode

1. ReadWriteOnce：普通磁盘
2. ReadOnlyMany：只读盘
3. ReadWriteMany：共享盘，包括NFS等

在Deployment的部署中，由于pod的数量不固定，Volume一般采用ReadOnlyMany或ReadWriteMany的模式。下面是一个例子，通过ReadOnly的Disk，在Deployment中部署ReadOnlyMany的PVC：

- 创建Disk，并把Disk挂载到VM上进行格式化

```
#创建Disk
gcloud compute disks create my-test-disk --zone us-central1-c

#挂载到VM上，格式化Disk
```

```
mkdir /test
mkfs.ext4 -m 0 -E lazy_itable_init=0,lazy_journal_init=0,discard /dev/sdb
mount -o discard,defaults /dev/sdb /test
echo 'Hello World!' > /test/index.html
umount /test

#将Disk从VM上卸载下来
```

- 创建PV和PVC

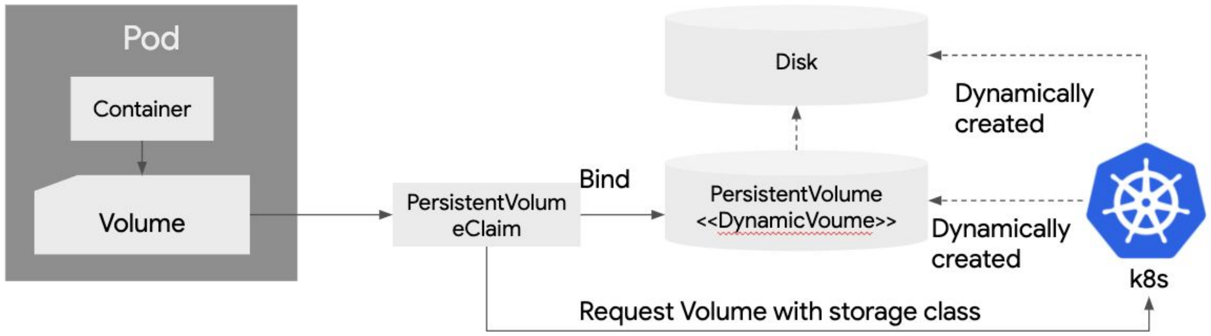
```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: my-readonly-pv
spec:
  storageClassName: slow
  capacity:
    storage: 10Gi
  accessModes:
    - ReadOnlyMany
  claimRef:
    namespace: default
    name: my-readonly-pvc
  gcePersistentDisk:
    pdName: my-test-disk
    fsType: ext4
    readOnly: true
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: my-readonly-pvc
spec:
  storageClassName: slow
  accessModes:
    - ReadOnlyMany
  resources:
    requests:
      storage: 10Gi
```

- Deployment

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 4
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        volumeMounts:
        - mountPath: /usr/share/nginx/html
          name: readonly-volume
          readOnly: true
        ports:
        - containerPort: 80
      volumes:
      - name: readonly-volume
        persistentVolumeClaim:
          claimName: my-readonly-pvc
          readOnly: true
```

2.1.2 动态部署

动态部署的方式可以由系统动态的部署pv和disk。大大方便pvc的部署。具体过程如下图：



Instead of admin to create PV manually, admin can deploy Persistent volume provisioner and define one or more storage class object to let users choose type of PV.

2.1.2.1 Pod应用pvc的动态部署

用户只需要部署pvc，就可以通过kubernetes自动创建disk和pv：

- 创建PVC，这里的storageClassName选择fast，在创建了PVC时，系统会根据sc的名字，自动选择相对应的disk类型，创建disk

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: myclaim
spec:
  storageClassName: fast
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 30Gi
```

- 创建Pod，引用PVC

```
apiVersion: v1
kind: Pod
metadata:
  name: mypod
spec:
  containers:
    - name: myfrontend
      image: nginx
      volumeMounts:
      - mountPath: "/usr/share/nginx/html"
        name: mypd
  volumes:
    - name: mypd
      persistentVolumeClaim:
        claimName: myclaim
```

- 查看

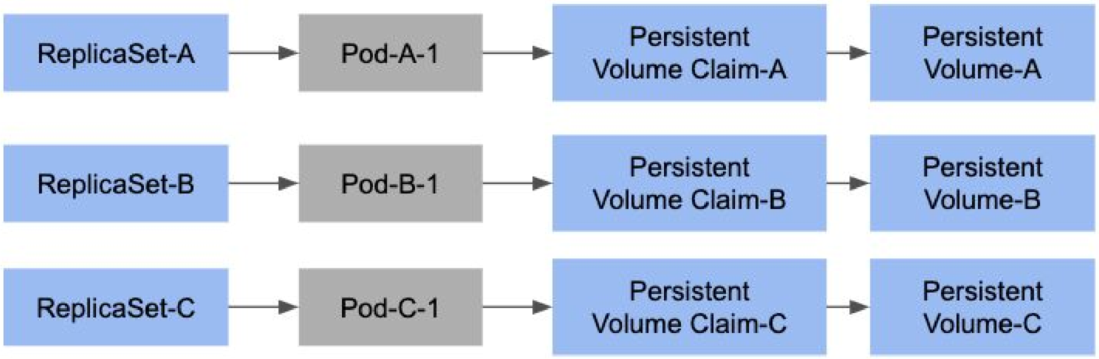
```
#查看pvc
$ kubectl get pvc
NAME      STATUS   VOLUME                                     CAPACITY
myclaim   Bound    pvc-28db65ac-9489-4323-a93b-130bc137d3bc   30Gi

ACCESS MODES   STORAGECLASS   AGE
RWO            fast           8h

#查看动态生成的pv
$ kubectl get pv
NAME                                       CAPACITY   ACCESS MODES
pvc-28db65ac-9489-4323-a93b-130bc137d3bc   30Gi       RWO

RECLAIM POLICY   STATUS   CLAIM             STORAGECLASS   REASON   AGE
Delete           Bound    default/myclaim   fast                    8h

#查看动态生成的disk
$ gcloud compute disks list
NAME
gke-cluster-1-d703c03e-pvc-28db65ac-9489-4323-a93b-130bc137d3bc
```

```

LOCATION       LOCATION_SCOPE  SIZE_GB  TYPE         STATUS
us-central1-c  zone            30       pd-ssd  READY
```

通过动态部署的方式非常方便的实现Pod获取disk的存储资源。

2.1.2.2 StatefulSet应用pvc的动态部署

在StatefulSet中，可以定义Volume的template，批量的动态创建pvc。这样每个StatefulSet中的Pod可以应用动态创建的disk。如下图：



- StatefulSet中引用volumeClaimTemplates：

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: web
spec:
  selector:
    matchLabels:
      app: nginx
  serviceName: "nginx"
  replicas: 3
  template:
    metadata:
      labels:
        app: nginx # has to match .spec.selector.matchLabels
    spec:
      terminationGracePeriodSeconds: 10
      containers:
      - name: nginx
        image: k8s.gcr.io/nginx-slim:0.8
        ports:
        - containerPort: 80
          name: web
        volumeMounts:
        - name: www
          mountPath: /usr/share/nginx/html
  volumeClaimTemplates:
  - metadata:
      name: www
    spec:
      accessModes: [ "ReadWriteOnce" ]
      storageClassName: slow
      resources:
        requests:
          storage: 1Gi
```

- 查看

```
$ kubectl get pvc
NAME         STATUS   VOLUME                                     CAPACITY   ACCESS
www-web-0    Bound    pvc-1fc70009-9a8b-4197-a678-98786ad38bdb   1Gi        RWO
www-web-1    Bound    pvc-ef9f1874-b00f-4f27-8556-a60992b852a1   1Gi        RWO
www-web-2    Bound    pvc-33f59629-bce4-4a87-9a02-a867597105b6   1Gi        RWO

MODES   STORAGECLASS   AGE
        slow           59s
        slow           43s
        slow           27s

$ kubectl get pv
NAME                                       CAPACITY   ACCESS MODES   RECLAIM POLICY
pvc-1fc70009-9a8b-4197-a678-98786ad38bdb   1Gi        RWO            Delete
pvc-33f59629-bce4-4a87-9a02-a867597105b6   1Gi        RWO            Delete
pvc-ef9f1874-b00f-4f27-8556-a60992b852a1   1Gi        RWO            Delete

STATUS   CLAIM               STORAGECLASS   REASON   AGE
Bound    default/www-web-0   slow                    63s
Bound    default/www-web-2   slow                    31s
Bound    default/www-web-1   slow                    48s

$ gcloud compute disks list
NAME                                                               LOCATION
gke-cluster-1-d703c03e-pvc-1fc70009-9a8b-4197-a678-98786ad38bdb  us-central1-c
gke-cluster-1-d703c03e-pvc-33f59629-bce4-4a87-9a02-a867597105b6  us-central1-c
gke-cluster-1-d703c03e-pvc-ef9f1874-b00f-4f27-8556-a60992b852a1  us-central1-c

LOCATION_SCOPE  SIZE_GB  TYPE         STATUS
zone            1        pd-standard  READY
zone            1        pd-standard  READY
zone            1        pd-standard  READY
```

StatefulSet动态的创建了PVC/PV和Disk。
在缩放pod或删除StatefulSet后，这些PVC依然保留，在恢复了pod后，这些Disk依旧挂载到相应的Pod上。

2.1.2.3 Replicas为1的Deployment采用pvc的动态部署

和前面类似，创建pvc，会自动创建pv和Disk

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc-persistent-cfg
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 50Gi
  storageClassName: slow
```

创建Deployment，Replicas为1，在volumes中应用pvc：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app
  labels:
    app: my-app
spec:
  replicas: 1
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
      - name: my-app
        image: nginx
        volumeMounts:
          - name: pv-data
            mountPath: /data
      volumes:
        - name: pv-data
          persistentVolumeClaim:
            claimName: pvc-persistent-cfg
```

由于Replicas为1，不会造成多读写的问题。

2.2 NFS

NFS的provisioner目前不是Kubernetes的官方支持的provisioner。可以通过helm进行安装。

1. stable/nfs-server-provisioner是提供nfs服务的provisioner
2. stable/nfs-client-provisioner是对已有的nfs提供provisioner

2.2.1 Deployment应用nfs-client静态部署

在已经安装过heml的Kubernetes集群上运行下面的命令：

```
helm install --set nfs.server=10.161.227.98 \
    --set nfs.path=/whgke stable/nfs-client-provisioner
```

```
NAME:   hopping-manatee
LAST DEPLOYED: Sat May 23 07:28:27 2020
NAMESPACE: default
STATUS: DEPLOYED
RESOURCES:
==> v1/ClusterRole
NAME                                           AGE
hopping-manatee-nfs-client-provisioner-runner  1s
==> v1/ClusterRoleBinding
NAME                                        AGE
run-hopping-manatee-nfs-client-provisioner  1s
==> v1/Deployment
NAME                                    READY  UP-TO-DATE  AVAILABLE  AGE
hopping-manatee-nfs-client-provisioner  0/1    1           0          1s
==> v1/Pod(related)
NAME                                               READY  STATUS             RESTARTS  AGE
hopping-manatee-nfs-client-provisioner-7..5-bkkqd  0/1    ContainerCreating  0         1s
apiVersion: v1
==> v1/Role
NAME                                                   AGE
leader-locking-hopping-manatee-nfs-client-provisioner  1s
==> v1/RoleBinding
NAME                                                   AGE
leader-locking-hopping-manatee-nfs-client-provisioner  1s
==> v1/ServiceAccount
NAME                                    SECRETS  AGE
hopping-manatee-nfs-client-provisioner  1        0s

==> v1/StorageClass
NAME        PROVISIONER                                           RECLAIMPOLICY
nfs-client  cluster.local/hopping-manatee-nfs-client-provisioner  Delete

VOLUMEBINDINGMODE  ALLOWVOLUMEEXPANSION  AGE
Immediate          true                  0s
```

可以观察到，安装了相应的SA/Binding，以及Provisioner和Storage Class: nfs-client。

- 创建静态的PV

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: nfs-pv
  labels:
    type: nfs
spec:
  storageClassName: nfs-client
  capacity:
    storage: 1Ti
  accessModes:
    - ReadWriteMany
  persistentVolumeReclaimPolicy: Retain
  nfs:
    path: /whgke
    server: 10.161.227.98
    readOnly: false
```

- 创建PVC

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: nfs-pvc
spec:
  storageClassName: "nfs-client"
  accessModes:
  - ReadWriteMany
  resources:
     requests:
       storage: 1Ti
  selector:
    matchLabels:
      type: nfs
```

- 创建Deployment

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app
  labels:
    app: my-app
spec:
  replicas: 3
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
      - name: my-app
        image: nginx
        volumeMounts:
          - name: pv-data
            mountPath: /usr/share/nginx/html
      volumes:
        - name: pv-data
          persistentVolumeClaim:
            claimName: nfs-pvc
```

这样，这个nfs被这个deployment中的pod

2.2.2 通过nfs-server提供集群内的nfs pvc

使用helm安装nfs-server-provisioner：

```
helm install stable/nfs-server-provisioner --name my-release
```

部署中的提示：

```
NAME:   my-release
LAST DEPLOYED: Sat May 23 07:57:36 2020
NAMESPACE: default
STATUS: DEPLOYED
RESOURCES:
==> v1/ClusterRole
NAME                               AGE
my-release-nfs-server-provisioner  1s
==> v1/ClusterRoleBinding
NAME                               AGE
my-release-nfs-server-provisioner  1s
==> v1/Pod(related)
NAME                                 READY  STATUS             RESTARTS  AGE
my-release-nfs-server-provisioner-0  0/1    ContainerCreating  0         1s
==> v1/Service
NAME                               TYPE       CLUSTER-IP  EXTERNAL-IP  PORT(S)
                     AGE
my-release-nfs-server-provisioner  ClusterIP  10.48.6.89  <none>
2049/TCP,2049/UDP,32803/TCP,32803/UDP,20048/TCP,20048/UDP,875/TCP,875/UDP,111/TCP,111/
UDP,662/TCP,662/UDP  1s
==> v1/ServiceAccount
NAME                               SECRETS  AGE
my-release-nfs-server-provisioner  1        1s
==> v1/StatefulSet
NAME                               READY  AGE
my-release-nfs-server-provisioner  0/1    1s
==> v1/StorageClass
NAME  PROVISIONER                                      RECLAIMPOLICY  VOLUMEBINDINGMODE
ALLOWVOLUMEEXPANSION  AGE

nfs   cluster.local/my-release-nfs-server-provisioner  Delete         Immediate          true
1s
kind: PersistentVolumeClaim
NOTES:
The NFS Provisioner service has now been installed.
A storage class named 'nfs' has now been created
and is available to provision dynamic volumes.
You can use this storageclass by creating a `PersistentVolumeClaim` with the
correct storageClassName attribute. For example:
    ---
    kind: PersistentVolumeClaim
    apiVersion: v1
    metadata:
      name: test-dynamic-volume-claim
    spec:
      storageClassName: "nfs"
      accessModes:
        - ReadWriteOnce
apiVersion: v1
      resources:
        requests:
          storage: 100Mi
```

创建基于nfs的pvc：

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: test-dynamic-volume-claim
spec:
  storageClassName: "nfs"
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 500Mi
```

创建deployment，采用此pvc:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app
  labels:
    app: my-app
spec:
  replicas: 3
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
      - name: my-app
        image: nginx
        volumeMounts:
          - name: pv-data
            mountPath: /usr/share/nginx/html
      volumes:
        - name: pv-data
          persistentVolumeClaim:
            claimName: test-dynamic-volume-claim
```

可以在my-app的pod中看到挂载的情况：

```
# df -h
Filesystem                               Size  Used Avail Use% Mounted on
10.48.6.89:/export/pvc-d6c2988a-...c09   95G   3.3G   92G   4% /usr/share/nginx/html
```

3 GCE-PD CSI Driver

Container Storage Interface（CSI）是容器存储接口的规范，用于管理用于存储数据的基于块和基于文件的卷。每个存储供应商都可以创建容器编排器一起使用的CSI驱动程序。 CSI驱动程序可以使用它来与Kubernetes控制器进行接口，用于管理persistent volume。CSI在Kubernetes 1.16支持Beta，Google也有相应CSI。具体部署方式如下：

- 下载

```
git clone \
  https://github.com/kubernetes-sigs/gcp-compute-persistent-disk-csi-driver
```

- 部署

```
export GCE_PD_SA_DIR=/home/hengwei/
export GCE_PD_SA_NAME=wh-service02.json
export PROJECT_ID=wh-service02
export GCE_PD_SA_NAME=k8scsi
./deploy/setup-project.sh
./deploy/kubernetes/deploy-driver.sh
```

- 创建csi-gce-pd的storage class

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: csi-gce-pd
provisioner: pd.csi.storage.gke.io
parameters:
  type: pd-standard
volumeBindingMode: WaitForFirstConsumer
```

- 查看

```
$ kubectl get sc
NAME       PROVISIONER           RECLAIMPOLICY VOLUMEBINDINGMODE    ALLOWVOLUMEEXPANSION AGE
csi-gce-pd pd.csi.storage.gke.io Delete        WaitForFirstConsumer false                4s
```

- 创建pvc和pod

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: podpvc
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: csi-gce-pd
  resources:
    requests:
      storage: 200Gi
---
apiVersion: v1
kind: Pod
metadata:
  name: web-server
spec:
  containers:
   - name: web-server
     image: nginx
     volumeMounts:
       - mountPath: /var/lib/www/html
         name: mypvc-csi
  volumes:
   - name: mypvc-csi
     persistentVolumeClaim:
       claimName: podpvc
       readOnly: false
```

- 查看

```
$ kubectl get pv
NAME                                       CAPACITY   ACCESS MODES
pvc-407b1f27-e4af-4964-ac2f-3f1b1e7a95f6   200Gi      RWO            Delete

RECLAIM POLICY   STATUS      CLAIM            STORAGECLASS   REASON   AGE
Bound       default/podpvc   csi-gce-pd              9s

$ kubectl get pvc
NAME      STATUS    VOLUME                                     CAPACITY
podpvc    Bound     pvc-407b1f27-e4af-4964-ac2f-3f1b1e7a95f6   200Gi

ACCESS MODES   STORAGECLASS   AGE
RWO            csi-gce-pd     23s
```

- Pod

```
$ kubectl describe pod web-server
Name:         web-server
...
Containers:
  web-server:
    ...
    Mounts:
      /var/lib/www/html from mypvc-csi (rw)
...
Volumes:
  mypvc-csi:
    Type:       PersistentVolumeClaim
    ClaimName:  podpvc
```

四 总结

在GKE上可以通过Volume的方式挂载Disk或NFS的存储资源。可以通过直接挂载的方式，也可以通过persisitentVolume的方式静态或动态的挂载。今后在Kubernetes中，CSI会成为Volume的主要的实现方式。因此在最后也介绍了如何部署和使用GCE-PD的CSI Diver。