NFS in Kubernetes

External Server Example

The usual patterns should work here:

- Provision a VM outside of Kubernetes/Terraform (e.g. `Ubuntu 16.04 LTS`)
- SSH in and install the `nfs-common` OS package (e.g. `sudo apt-get update && sudo apt-get -y install nfs-common`)
- Create your `/exports` and run an NFS server
- Open ports `2049`, `20048`, and `111` firewall using OpenStack security groups
- Consume the NFS mount from Kubernetes

Consuming the Mount

```
volumes:
- name: nfs
  nfs:
    server: <NFS_SERVER_IP>
    path: /
```

Dynamic Volumes with the NFS Client Provisioner

For an easy way to get up and running serving shared volumes on Kubernetes from an ***existing*** NFS server, check out the nfs-client provisioner.

Provisioners like this one allow you to use Kubernetes to manage the lifecycle of your Kubernetes volumes and their data using PVCs, or `PersistentVolumeClaims`.

The basic workflow is as follows:

1. User creates a PVC resource in Kubernetes with an attached `StorageClass` (if none is specified, the default will be used based on your cloud type - this blog post lists the default storage classes for each cloud provider)
2. Kubernetes uses the StorageClass to determine how/whether a PV (`PersistentVolume`) should be created based on the claim parameters (e.g. for NFS, this effectively does a `mkdir`)
3. If no existing static PV matches the parameters in the PVC, a new one should be dynamically created
4. The PV can then be mounted into client pods by specifying the PVC name as a volume

For more details, see https://kubernetes.io/docs/concepts/storage/persistent-volumes/#dynamic

In-Cluster Server Example

Following this example, I was able to easily get an NFS server pod running within a Kubernetes 1.9 cluster.

The original example was intended for use with GCE and included some nice features like backing your NFS server with PVCs.

Prerequisites

- SSH in to each node and install the `nfs-common` OS package (e.g. `sudo apt-get update && sudo apt-get -y install nfs-common`)

Ideally, this could be handled by the Terraform process, but may change depending on the VM image specified (e.g. `apt` vs `yum` vs `dnf` vs `apk`, etc

Specific Modifications

The StorageClasses are specific to GCE, so we do not need to include those. Instead, we rely on our Terraform process mounting an external volume to the storage nodes.

Modifying the above example slightly to work with our Terraform/hostPath process, we have the following set of YAMLs:

nfs-server-rc.yaml

```yaml
kind: Service
apiVersion: v1
metadata:
  name: nfs-server
spec:
  ports:
    - name: nfs
      port: 2049
    - name: mountd
      port: 20048
    - name: rpcbind
      port: 111
  selector:
    role: nfs-server
---
apiVersion: v1
kind: ReplicationController
metadata:
  name: nfs-server
spec:
  replicas: 1
  selector:
    role: nfs-server
  template:
    metadata:
      labels:
        role: nfs-server
    spec:
      nodeSelector:
        external-storage: "true"
      containers:
      - name: nfs-server
        image: gcr.io/google_containers/volume-nfs:0.8
        ports:
          - name: nfs
            containerPort: 2049
          - name: mountd
            containerPort: 20048
          - name: rpcbind
            containerPort: 111
        securityContext:
          privileged: true
        volumeMounts:
          - mountPath: /exports
            name: nfs-export-fast
      volumes:
        - name: nfs-export-fast
          hostPath:
            path: /data/nfs
```

Consuming the Mount

The following example Pod consumes our in-cluster NFS export.

Note that the `server` field is populated with the Kubernetes Service IP (e.g. `kubectl get svc`):

web-pod.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: web
spec:
  containers:
  - name: web
    image: nginx
    volumeMounts:
      # name must match the volume name below
    - name: nfs
      mountPath: "/usr/share/nginx/html/"
    ports:
    - name: web
      containerPort: 80
      protocol: TCP
  volumes:
  - name: nfs
    nfs:
      # FIXME: use the right name
      #server: nfs-server.default.kube.local
      server: "10.101.9.169"
      path: "/"
      readOnly: false
```

Testing it Out

After creating our NFS server and a pod consuming it, we can use `kubectl exec` to test that our NFS is working as expected:

```
# Check the node and name of our web pod
ubuntu@mltf-master:~$ kubectl get pods
NAME               READY     STATUS    RESTARTS   AGE       IP           NODE
nfs-server-wc8h6   1/1       Running   0          21m       10.244.1.4   mltf-storage0
web                1/1       Running   0          11m       10.244.3.2   mltf-storage1

# Exec into the container to test writing to the NFS
ubuntu@mltf-master:~$ kubectl exec -it web -- bash
root@web:/# cat /usr/share/nginx/html/index.html
Hello from NFS!

# Test writing to a file
root@web:/# vi /usr/share/nginx/html/pod-write
bash: vi: command not found

# No "vi" included in nginx image, so we install it
root@web:/# apt-get update && apt-get install vim
Ign:1 http://cdn-fastly.deb.debian.org/debian stretch InRelease
Get:2 http://security.debian.org/debian-security stretch/updates InRelease [94.3 kB]
    ...    ...    ...    ...    ...    ...    ...    ...    ...
Reading state information... Done
The following additional packages will be installed:
  libgpm2 vim-common vim-runtime xxd
Suggested packages:
  gpm ctags vim-doc vim-scripts
The following NEW packages will be installed:
  libgpm2 vim vim-common vim-runtime xxd
0 upgraded, 5 newly installed, 0 to remove and 2 not upgraded.
Need to get 6766 kB of archives.
After this operation, 31.2 MB of additional disk space will be used.
Do you want to continue? [Y/n]
Get:1 http://cdn-fastly.deb.debian.org/debian stretch/main amd64 xxd amd64 2:8.0.0197-4+deb9u1 [132 kB]
Get:2 http://cdn-fastly.deb.debian.org/debian stretch/main amd64 vim-common all 2:8.0.0197-4+deb9u1 [159 kB]
    ...    ...    ...    ...    ...    ...    ...    ...    ...
update-alternatives: warning: skip creation of /usr/share/man/ja/man1/editor.1.gz because associated file /usr
/share/man/ja/man1/vim.1.gz (of link group editor) doesn't exist
update-alternatives: warning: skip creation of /usr/share/man/man1/editor.1.gz because associated file /usr
/share/man/man1/vim.1.gz (of link group editor) doesn't exist

# Test writing to a file
root@web:/# vi /usr/share/nginx/html/pod-write
root@web:/# cat /usr/share/nginx/html/pod-write
asdf 1234 Hello, World!
root@web:/# exit
exit

# SSH into "storage0", where our NFS server pod is running
ubuntu@mltf-master:~$ ssh 192.168.0.3
Welcome to Ubuntu 16.04.4 LTS (GNU/Linux 4.4.0-127-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  Get cloud support with Ubuntu Advantage Cloud Guest:
    http://www.ubuntu.com/business/services/cloud

14 packages can be updated.
7 updates are security updates.

Last login: Mon Jun 11 16:28:49 2018 from 192.168.0.6

# Verify that the file created inside of our web pod (running on "storage1") was persisted to the NFS directory
on "storage0":
ubuntu@mltf-storage0:~$ cat /data/nfs/pod-write
asdf 1234 Hello, World!
```

We can also ensure that these NFS volumes can be mounted into multiple pods simultaneously (e.g. ReadWriteMany):

```
# Create some duplicate pods consuming the same NFS mount
ubuntu@mltf-master:~/kubernetes-nfs-server$ kubectl create -f web-pod2.yaml -f web-pod3.yaml
pod "web2" created
pod "web3" created


# Wait for containers to start
ubuntu@mltf-master:~/kubernetes-nfs-server$ kubectl get pods -o wide
NAME               READY     STATUS              RESTARTS   AGE       IP           NODE
nfs-server-wc8h6   1/1       Running             0          49m       10.244.1.4   mltf-storage0
web                1/1       Running             0          39m       10.244.3.2   mltf-storage1
web2               0/1       ContainerCreating   0          8s        <none>       mltf-storage0
web3               0/1       ContainerCreating   0          8s        <none>       mltf-worker0


# Verify that the file created inside of our original web pod (running on "storage1") also shows up here
ubuntu@mltf-master:~/kubernetes-nfs-server$ kubectl exec -it web2 -- cat /usr/share/nginx/html/pod-write
asdf 1234 Hello, World!
ubuntu@mltf-master:~/kubernetes-nfs-server$ kubectl exec -it web3 -- cat /usr/share/nginx/html/pod-write
asdf 1234 Hello, World!
```

Dynamic Volumes using the NFS Provisioner

See https://github.com/kubernetes-incubator/external-storage/tree/master/nfs

In a slightly more complex example, we can actually have Kubernetes provision volumes dynamically for us from an NFS export.

1. Clone the repo above
2. Locate the nfs/deploy/kubernetes subfolder
3. Edit deployment.yaml (e.g. change nodeSelector, change mount path/config)
4. kubectl create -f auth/serviceaccount.yaml -f auth/clusterrole.yaml -f auth/clusterrolebinding.yaml
5. kubectl create -f class.yaml -f deployment.yaml

Congratulations! Wasn't that easy?

Testing it Out

A simple test case for consuming a dynamic PVC can be found below:

1. kubectl create -f claim.yaml
2. kubectl create -f write_pod.yaml

Expected output:

```
ubuntu@mltf-master:~/external-storage/nfs/deploy/kubernetes$ kubectl get pvc,pv
NAME                        STATUS    VOLUME                                     CAPACITY   ACCESS MODES
STORAGECLASS   AGE
persistentvolumeclaim/nfs   Bound     pvc-253140dc-6dc2-11e8-9b58-fa163eace583   1Mi        RWX
example-nfs    15m

NAME                                                       CAPACITY   ACCESS MODES   RECLAIM POLICY
STATUS    CLAIM         STORAGECLASS   REASON    AGE
persistentvolume/pvc-253140dc-6dc2-11e8-9b58-fa163eace583   1Mi        RWX            Delete
Bound     default/nfs   example-nfs              15m

# Look up the UID of our PVC (we will need this to find it on disk later)
ubuntu@mltf-master:~/external-storage/nfs/deploy/kubernetes$ kubectl get pvc -o yaml
apiVersion: v1
items:
- apiVersion: v1
  kind: PersistentVolumeClaim
  metadata:
    annotations:
      control-plane.alpha.kubernetes.io/leader: '{"holderIdentity":"c9839a10-6dc1-11e8-b790-0a580af40112","
leaseDurationSeconds":15,"acquireTime":"2018-06-11T21:55:25Z","renewTime":"2018-06-11T21:55:27Z","
leaderTransitions":0}'
    ...    ...    ...    ...    ...    ...    ...    ...    ...    ...
      volume.beta.kubernetes.io/storage-class: example-nfs
```

```
      volume.beta.kubernetes.io/storage-provisioner: example.com/nfs
    ...    ...    ...    ...    ...    ...    ...    ...    ...    ...
    uid: 253140dc-6dc2-11e8-9b58-fa163eace583
  spec:
    ...    ...    ...    ...    ...    ...    ...    ...    ...    ...
      requests:
        storage: 1Mi
    volumeName: pvc-253140dc-6dc2-11e8-9b58-fa163eace583
  status:
    ...    ...    ...    ...    ...    ...    ...    ...    ...    ...

# Wait for the write-pod to finish executing
ubuntu@mltf-master:~/external-storage/nfs/deploy/kubernetes$ kubectl get pods
NAME                                READY     STATUS      RESTARTS   AGE            NODE
mongo-k2pqv                         1/1       Running     0          1h   mltf-worker0
ndslabs-apiserver-pd62v             2/2       Running     0          1h   mltf-storage1
ndslabs-etcd-ppsdv                  1/1       Running     0          23m   mltf-worker0
ndslabs-webui-qx4fb                 1/1       Running     0          1h   mltf-storage1
nfs-provisioner-7f78fcc699-vjr8d    1/1       Running     0          6m   mltf-storage0
nfs-server-wc8h6                    1/1       Running     0          5h    mltf-storage0
web                                 1/1       Running     0          2h    mltf-storage0
web2                                1/1       Running     0          4h    mltf-storage0
web3                                1/1       Running     0          4h    mltf-worker0
write-pod                           0/1       Completed   0          2m    mltf-storage1    <-------

# SSH into "storage0" (this is where the nfs-provisioner is currently running)
ubuntu@mltf-master:~/external-storage/nfs/deploy/kubernetes$ ssh 192.168.0.3
Welcome to Ubuntu 16.04.4 LTS (GNU/Linux 4.4.0-127-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  Get cloud support with Ubuntu Advantage Cloud Guest:
    http://www.ubuntu.com/business/services/cloud

14 packages can be updated.
7 updates are security updates.


Last login: Mon Jun 11 21:55:53 2018 from 192.168.0.6

# Locate the subdirectory of the nfs-provisioner on our OpenStack volume for shared storage
ubuntu@mltf-storage0:~$ ls -al /data
total 32
drwxr-xr-x  5 root root  4096 Jun 11 21:51 .
drwxr-xr-x 26 root root  4096 Jun 11 21:19 ..
drwx------  2 root root 16384 Jun  8 22:46 lost+found
drwxr-xr-x  4 root root  4096 Jun 11 20:52 nfs
drwxr-xr-x  3 root root  4096 Jun 11 21:55 nfs-provisioner

# Locate the subdirectory of our specific PVC (NOTE: this will match the "volume name" and UID of the PVC we
looked up earlier)
ubuntu@mltf-storage0:~$ ls -al /data/nfs-provisioner/
total 36
drwxr-xr-x 3 root root  4096 Jun 11 21:55 .
drwxr-xr-x 5 root root  4096 Jun 11 21:51 ..
-rw-r--r-- 1 root root 14791 Jun 11 21:55 ganesha.log
-rw------- 1 root root    36 Jun 11 21:51 nfs-provisioner.identity
drwxrwsrwx 2 root root  4096 Jun 11 21:56 pvc-253140dc-6dc2-11e8-9b58-fa163eace583
-rw------- 1 root root   902 Jun 11 21:55 vfs.conf

# Verify the contents of the PVC
ubuntu@mltf-storage0:~$ ls -al /data/nfs-provisioner/pvc-253140dc-6dc2-11e8-9b58-fa163eace583/
total 8
drwxrwsrwx 2 root root 4096 Jun 11 21:56 .
drwxr-xr-x 3 root root 4096 Jun 11 21:55 ..
-rw-r--r-- 1 root root    0 Jun 11 21:56 SUCCESS     <------
```