GlusterFS HostHack in Kubernetes

Getting Started

Clone the GlusterFS repo containing the necessary Kubernetes specs:

```
git clone https://github.com/nds-org/gluster.git
cd gluster/
```

Server Setup

Create the gluster-server DaemonSet using *kubectl*:

```
kubectl create -f kubernetes/gluster-server-ds.yaml
```

This spec runs the ndslabs/gluster container in "server mode" on Kubernetes nodes labeled with **ndslabs-role=storage**.

Once all of the server containers are up, we must tell them to cooperate with each other using the *gluster* CLI.

The steps below then only to be done from inside of a single glusterfs-server container.

Alternative: Raw Docker

```
docker run --name=gfs --net=host --pid=host --privileged -v /dev:/dev -v <ABSOLUTE_PATH_TO_SHARED_DATA>:/var
/glfs -v /run:/run -v /:/media/host -it -d gluster:local
```

Getting into a Server Container

Using *kubectl*, exec into one of the GlusterFS servers:

```
core@willis8-k8-test-1 ~ $ kubectl get pods -o wide
NAME                        READY     STATUS    RESTARTS   AGE       NODE
coffee-rc-4u3pb             1/1       Running   0          12d       192.168.100.65
coffee-rc-5m4t6             1/1       Running   0          12d       192.168.100.65
default-http-backend-y98iw  1/1       Running   0          22h       192.168.100.64
glusterfs-server-hh5rm      1/1       Running   0          5d        192.168.100.156
glusterfs-server-zoefs      1/1       Running   0          5d        192.168.100.89
ndslabs-apiserver-zqgj8     1/1       Running   0          1d        192.168.100.66
ndslabs-gui-p0hjh           1/1       Running   0          23h       192.168.100.66
nginx-ilb-rc-x853y          1/1       Running   0          6d        192.168.100.64
tea-rc-8saiu                1/1       Running   0          12d       192.168.100.65
tea-rc-t403k                1/1       Running   0          12d       192.168.100.65
core@willis8-k8-test-1 ~ $ kubectl exec -it glusterfs-server-zoefs bash
```

Take note of all node IPs that are running glusterfs-server pods. You will need these IPs to finish configuring GlusterFS.

Peer Probe

Once inside of the gluster server container, perform a peer probe on all other gluster nodes.

Do not probe the host's own IP.

For example, since we are executing from 192.168.100.89, we must probe our other storage node:

```
root@willis-k8-test-gluster:/# gluster peer probe 192.168.100.156
```

Create Volume

Ansible has already created the placeholder directories for bricks, we just need to create and start a Gluster volume pointing to the different brick directories on each node.

This is done using *gluster create volume* as outlines below:

```
root@willis-k8-test-gluster:/# gluster volume create ndslabs transport tcp 192.168.100.89:/var/glfs/brick0
192.168.100.156:/var/glfs/ndslabs/brick0
```

NOTE: Our Ansible playbook mounts GlusterFS bricks at /media/brick0. We will need to update this in the future to be consistent throughout.

To be sure the volume was created successfully, you can run the following commands and see your new volume:

```
root@willis-k8-test-gluster:/# gluster volume list
ndslabs
root@willis-k8-test-gluster:/# gluster volume status
Volume ndslabs is not started
```

Reusing a Volume

Simply add **force** to the end of your **volume create** command to force GlusterFS to reuse a volume that is no longer accessible:

```
root@willis-k8-test-gluster:/# gluster volume create ndslabs transport tcp 192.168.100.89:/media/brick0/brick
/ndslabs 192.168.100.156:/media/brick0/brick/ndslabs
volume create: ndslabs: failed: /media/brick0/brick/ndslabs is already part of a volume
root@willis-k8-test-gluster:/# gluster volume create ndslabs transport tcp 192.168.100.89:/media/brick0/brick
/ndslabs 192.168.100.156:/media/brick0/brick/ndslabs force
volume create: ndslabs: success: please start the volume to access data
```

The alternative solution would be to delete / recreate the mount point:

```
root@willis-k8-test-gluster:/# rm -rf /path/to/brick0
root@willis-k8-test-gluster:/# mkdir -p /path/to/brick0
```

Start Volume

Now that we have created our volume, we must start it in order for clients to mount it:

```
root@willis-k8-test-gluster:/# gluster volume start ndslabs
volume start: ndslabs: success
```

Our volume is now being served out to the cluster over NFS, and we are ready for our clients to mount the volume.

Adding a Brick

Suppose we have a simple replicated gluster volume with 2 bricks, and we are running low on space... we want to expand the storage it contains:

```
# On the host node, via SSH
core@workshop1-node1 ~ $ df
Filesystem             1K-blocks     Used Available Use% Mounted on
devtmpfs                16460056        0  16460056   0% /dev
tmpfs                   16476132        0  16476132   0% /dev/shm
tmpfs                   16476132     1872  16474260   1% /run
tmpfs                   16476132        0  16476132   0% /sys/fs/cgroup
/dev/vda9               38216204   256716  36301140   1% /
/dev/mapper/usr          1007760   639352    316392  67% /usr
tmpfs                   16476132    17140  16458992   1% /tmp
tmpfs                   16476132        0  16476132   0% /media
/dev/vda1                 130798    39292     91506  31% /boot
/dev/vda6                 110576       64    101340   1% /usr/share/oem
/dev/vdb                41922560  6023596  35898964  15% /var/lib/docker
/dev/vdc                10475520   626268   9849252   6% /media/storage
/dev/vdd               104806400 49157880  55648520  47% /media/brick0
192.168.100.122:global 104806400 87618944  17187456  84% /var/glfs/global
tmpfs                    3295224        0   3295224   0% /run/user/500
/dev/vde               209612800    32928 209579872   1% /media/brick1


# Inside of the GLFS server pod
root@workshop1-node1:/# gluster volume info global

Volume Name: global
Type: Replicate
Volume ID: ca59a98e-c959-454e-8ac3-9082b0ed2856
Status: Started
Snapshot Count: 0
Number of Bricks: 1 x 2 = 2
Transport-type: tcp
Bricks:
Brick1: 192.168.100.122:/media/brick0/brick
Brick2: 192.168.100.116:/media/brick0/brick
Options Reconfigured:
nfs.disable: on
performance.readdir-ahead: on
transport.address-family: inet
features.quota: on
features.inode-quota: on
features.quota-deem-statfs: on
```

Provision and attach a new OpenStack volume to your existing instance, then format it with XFS:

```
core@workshop1-node1 ~ $ sudo mkfs -t xfs /dev/vde
meta-data=/dev/vde               isize=256    agcount=4, agsize=13107200 blks
         =                       sectsz=512   attr=2, projid32bit=1
         =                       crc=0        finobt=0
data     =                       bsize=4096   blocks=52428800, imaxpct=25
         =                       sunit=0      swidth=0 blks
naming   =version 2              bsize=4096   ascii-ci=0 ftype=0
log      =internal log           bsize=4096   blocks=25600, version=2
         =                       sectsz=512   sunit=0 blks, lazy-count=1
realtime =none                   extsz=4096   blocks=0, rtextents=0
```

You will then need to build up a *.mount file as below:

```
$ vi media-brick1.mount
[Unit]
Description=Mount OS_DEVICE on MOUNT_PATH
After=local-fs.target

[Mount]
What=OS_DEVICE
Where=MOUNT_PATH
Type=FS_TYPE
Options=noatime

[Install]
WantedBy=multi-user.target
```

where:

- **OS_DEVICE** is the source device in */dev* where your raw volume is mounted (i.e. */dev/vde*)
- **MOUNT_PATH** is the target mount path where your data should be mounted (i.e. */media/brick1*)
- **FS_TYPE** is a string of which filesystem will be formatted on the new volume (i.e. *xfs*)

Place this file in */etc/systemd/system/*

Finally, start and enable your service to mount the volume to CoreOS and ensure it is remounted on restart:

```
sudo mv media-brick1.mount /etc/systemd/system/media-brick1.mount
sudo systemctl daemon-reload
sudo systemctl start media-brick1.mount
sudo systemctl enable  media-brick1.mount
sudo systemctl unmask  media-brick1.mount
```

You will need to perform the above steps on each of your GLFS servers before continuing

Now you'll need to exec into one of the GLFS server pods and perform the following:

```
# Peer probe the other IP in the cluster (gluster service IP also seems to work)
$ gluster peer probe 10.254.202.236
peer probe: success. Host 192.168.100.1 port 24007 already in peer list


# This one fails because we did not include our new brick's second replica
$ gluster volume add-brick global 192.168.100.122:/media
/brick1
volume add-brick: failed: Incorrect number of bricks supplied 1 with count 2


# This one fails because we need a sub-directory of the mount point
$ gluster volume add-brick global 192.168.100.122:/media/brick1 192.168.100.116:/media/brick1
volume add-brick: failed: The brick 192.168.100.116:/media/brick1 is a mount point. Please create a sub-
directory under the mount point and use that as the brick directory. Or use 'force' at the end of the command
if you want to override this behavior.


# This one works! :D
$ gluster volume add-brick global 192.168.100.122:/media/brick1/brick 192.168.100.116:/media/brick1/brick
volume add-brick: success
```

And now we can see that our new brick has been added to the existing volume:

```
core@workshop1-node1 ~ $ df
Filesystem             1K-blocks     Used Available Use% Mounted on
devtmpfs                16460056        0  16460056   0% /dev
tmpfs                   16476132        0  16476132   0% /dev/shm
tmpfs                   16476132     1792  16474340   1% /run
tmpfs                   16476132        0  16476132   0% /sys/fs/cgroup
/dev/vda9               38216204   256736  36301120   1% /
/dev/mapper/usr          1007760   639352    316392  67% /usr
tmpfs                   16476132    17140  16458992   1% /tmp
tmpfs                   16476132        0  16476132   0% /media
/dev/vda1                 130798    39292     91506  31% /boot
/dev/vda6                 110576       64    101340   1% /usr/share/oem
/dev/vdb                41922560  6023732  35898828  15% /var/lib/docker
/dev/vdc                10475520   626360   9849160   6% /media/storage
/dev/vdd               104806400 49157820  55648580  47% /media/brick0
192.168.100.122:global 314419200 49191424 265227776  16% /var/glfs/global
tmpfs                    3295224        0   3295224   0% /run/user/500
/dev/vde               209612800    33088 209579712   1% /media/brick1


root@workshop1-node1:/# gluster volume info global

Volume Name: global
Type: Distributed-Replicate
Volume ID: ca59a98e-c959-454e-8ac3-9082b0ed2856
Status: Started
Snapshot Count: 0
Number of Bricks: 2 x 2 = 4
Transport-type: tcp
Bricks:
Brick1: 192.168.100.122:/media/brick0/brick
Brick2: 192.168.100.116:/media/brick0/brick
Brick3: 192.168.100.122:/media/brick1/brick
Brick4: 192.168.100.116:/media/brick1/brick
Options Reconfigured:
nfs.disable: on
performance.readdir-ahead: on
transport.address-family: inet
features.quota: on
features.inode-quota: on
features.quota-deem-statfs: on
```

Client Setup

Create the gluster-client DaemonSet using *kubectl*:

```
kubectl create -f kubernetes/gluster-client-ds.yaml
```

This spec runs the ndslabs/gluster container in "client mode" on Kubernetes nodes labeled with **ndslabs-role=compute**.

Once each client container starts, it will mount the GlusterFS volume to each **compute** host using NFS.

Testing

Once the clients are online, we can run a simple test of GlusterFS to ensure that it is correctly serving and synchronizing the volume.

From the Kubernetes master, run the following command to see which nodes are running the glusterfs-client containers:

```
core@willis8-k8-test-1 ~ $ kubectl get pods -o wide
NAME                          READY     STATUS    RESTARTS   AGE       NODE
coffee-rc-4u3pb               1/1       Running   0          12d       192.168.100.65
coffee-rc-5m4t6               1/1       Running   0          12d       192.168.100.65
default-http-backend-y98iw    1/1       Running   0          23h       192.168.100.64
glusterfs-client-4hm9y        1/1       Running   0          5d        192.168.100.65
glusterfs-client-6c12y        1/1       Running   0          5d        192.168.100.66
glusterfs-server-hh5rm        1/1       Running   0          5d        192.168.100.156
glusterfs-server-zoefs        1/1       Running   0          5d        192.168.100.89
ndslabs-apiserver-zqgj8       1/1       Running   0          1d        192.168.100.66
ndslabs-gui-p0hjh             1/1       Running   0          23h       192.168.100.66
nginx-ilb-rc-x853y            1/1       Running   0          6d        192.168.100.64
tea-rc-8saiu                  1/1       Running   0          12d       192.168.100.65
tea-rc-t403k                  1/1       Running   0          12d       192.168.100.65
```

Create two SSH sessions - one into each compute node (in this case, 192.168.100.65 and 192.168.100.66).

First Session

In one SSH session, run a BusyBox image mounted with our shared volume:

```
docker run -v /var/glfs:/var/glfs --rm -it busybox
```

Inside of the BusyBox container, create a test file:

```
echo "testing!" > /var/glfs/ndslabs/test.file
```

Second Session

On the other machine, test that mapping the same directory into BusyBox we can see the changes from the first host:

```
docker run -v /var/glfs:/var/glfs --rm -it busybox
```

Running an *ls* on */var/glfs/ndslabs/* should show the test file created on the other node:

```
ls -al /var/glfs/ndslabs
```

This proves that we can mount via NFS onto each node, map the NFS mount into containers, and allow those containers to ingest or modify the data from the NFS mount.