GlusterFS in Kubernetes

The purpose of this document is to familiarize you with running GlusterFS under Kubernetes.

GlusterFS is an open-source distributed filesystem that allows for PVCs that support ReadWriteMany.

Overview

Running GlusterFS in Kubernetes with PVC support is easier than ever with the GlusterFS Simple Provisioner!

Prerequisites

- 2+ node Kubernetes cluster
- No PVC support currently installed

The Long Way

The external-storage repo gives instructions for bringing this all up by hand.

First Steps

First, you'll need to clone the external-storage repo from the kubernetes-incubator:

```
$ git clone https://github.com/kubernetes-incubator/external-storage && cd external-storage
```

Locate the gluster/glusterfs subdirectory, which contains these same instructions on getting things up and running:

```
$ cd gluster/glusterfs/
```

Apply the correct node label to each of your storage nodes:

```
$ kubectl label nodes <storage-node-name> storagenode=glusterfs
node/<storage-node-name> labeled
```

Start GlusterFS

Bring up the GlusterFS DaemonSet and wait for them to come online:

```
$ kubectl create -f deploy/glusterfs-daemonset.yaml
daemonset.extensions/glusterfs created


$ kubectl get pods -l glusterfs-node=pod --watch
```

Locate your pod IPs once they are online:

```
$ kubectl get pods -o wide | grep glusterfs | grep -v provisioner
NAME                                        READY     STATUS    RESTARTS   AGE       IP
NODE              NOMINATED NODE
glusterfs-t44m5                             1/1       Running   0          4h        192.168.0.9   nfstest-
storage1   <none>
glusterfs-v64wn                             1/1       Running   0          4h        192.168.0.4   nfstest-
storage0   <none>


$ kubectl get pods -o wide | grep glusterfs | grep -v provisioner | awk '{print $6}'
192.168.0.9
192.168.0.4
```

Exec into each glusterfs pod and perform a `gluster peer probe` on **the other pod's IP**:

```
$ kubectl exec -it glusterfs-t44m5 -- gluster peer probe 192.168.0.4
peer probe: success.


$ kubectl exec -it glusterfs-v64wn -- gluster peer probe 192.168.0.9
peer probe: success. Host 192.168.0.9 port 24007 already in peer list
```

Congratulations! You now have a GlusterFS cluster running on top of Kubernetes!

Start GlusterFS Simple Provisioner

To install PVC support on your GlusterFS, you need to build up a custom *StorageClass* containing your GlusterFS pod IPs.

This will also require you to choose a "brick path", a directory on the host where your gluster bricks should be housed.

Normally this path would be mounted from an external volume, but for this example we are just using `/tmp` (NOTE: this is obviously not advised in production, as `/tmp` is typically cleared upon restart):

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
  name: glusterfs-simple
parameters:
  brickrootPaths: "192.168.0.9:/tmp,192.168.0.4:/tmp"
  forceCreate: "true"
  volumeType: "replica 2"
provisioner: gluster.org/glusterfs-simple
```

For Kubernetes 1.8+, you will also need to install RBAC permissions for the provisioner:

```
$ kubectl create -f deploy/rbac.yaml
serviceaccount/glfs-provisioner created
clusterrole.rbac.authorization.k8s.io/glfs-provisioner-runner created
clusterrolebinding.rbac.authorization.k8s.io/run-glfs-provisioner created
```

You are now ready to run the GlusterFS Simple Provisioner deployment:

```
$ kubectl create -f deploy/deployment.yaml
deployment.extensions/glusterfs-simple-provisioner created
```

The Short Way

Execute the following bash script from your Kubernetes master node to set everything up for you:

```
$ chmod +x ./deploy-glfs.sh
$ ./deploy-glfs <number_of_storage_nodes>
```

deploy-glfs.sh

```
#!/bin/bash
#
# Usage: ./deploy-glfs.sh <number_of_storage_nodes>
#

# DEBUG ONLY: Set this to "echo" to neuter the script and perform a dry-run
DEBUG=""

# The host directory to store brick files
BRICK_HOSTDIR="/tmp"

# Read in the desired number of storage nodes from first arg
NODE_COUNT="$1"

# Ensure that we have enough storage nodes to run GLFS
if [ "$NODE_COUNT" -lt 2 ]; then
  echo "ERROR: Cannot deploy GlusterFS with less than 2 nodes"
  exit 1
fi

# Clone external-storage repo for NFS provisioner templates
$DEBUG git clone https://github.com/kubernetes-incubator/external-storage

# Label storage nodes appropriately
STORAGE_NODES=$(kubectl get nodes --no-headers | grep storage | awk '{print $1}')
for node in $STORAGE_NODES; do
  $DEBUG kubectl label nodes $node storagenode=glusterfs
done

# Create the GLFS cluster
$DEBUG kubectl apply -f external-storage/gluster/glusterfs/deploy/glusterfs-daemonset.yaml

# Wait for the GLFS cluster to come up
count="$(kubectl get pods --no-headers | grep glusterfs | grep -v provisioner | awk '{print $3}' | grep Running
| wc -l)"
while [ "$count" -lt "$NODE_COUNT" ]; do
  echo "Waiting for GLFS: $count / $NODE_COUNT"
  sleep 5
  count="$(kubectl get pods --no-headers | grep glusterfs | grep -v provisioner | sed -e s/[\\n\\r]//g | awk
'{print $3}' | grep -o Running | wc -l)"
done
echo "GlusterFS is now Running: $count / $NODE_COUNT"

# Retrieve GlusterFS pod IPs
PEER_IPS=$(kubectl get pods -o wide | grep glusterfs | grep -v provisioner | awk '{print $6}')

# Use pod names / IPs to exec in and perform `gluster peer probe`
for pod_ip in ${PEER_IPS}; do
  for peer_ip in ${PEER_IPS}; do
    # Skip each node probing itself
    if [ "$pod_ip" == "$peer_ip" ]; then
      continue;
    fi

    # Perform a gluster peer probe
    pod_name=$(kubectl get pods -o wide | grep $pod_ip | awk '{print $1}')
    $DEBUG kubectl exec -it $pod_name gluster peer probe $peer_ip
  done;
done;
```

```
# Dynamically build StorageClass from pod IPs (see below)
BRICK_PATHS=""
for pod_ip in ${PEER_IPS[@]}; do
  # Insert comma if we already started accumlating ips/paths
  if [ "$BRICK_PATHS" != "" ]; then
    BRICK_PATHS="$BRICK_PATHS,"
  fi

  # Build up brickrootPaths one host at a time
  BRICK_PATHS="${BRICK_PATHS}${pod_ip}:${BRICK_HOSTDIR}"
done

# Modify StorageClass to contain our GlusterFS brickrootPaths
echo "---
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: glusterfs-simple
provisioner: gluster.org/glusterfs-simple
parameters:
  forceCreate: \"true\"
  volumeType: \"replica 2\"
  brickrootPaths: \"$BRICK_PATHS\"
" > external-storage/gluster/glusterfs/deploy/storageclass.yaml

# Create the storage class
$DEBUG kubectl apply -f external-storage/gluster/glusterfs/deploy/storageclass.yaml

# Bind the necessary ServiceAccount / ClusterRole
$DEBUG kubectl apply -f external-storage/gluster/glusterfs/deploy/rbac.yaml

# Create the GLFS Simple Provisioner
$DEBUG kubectl apply -f external-storage/gluster/glusterfs/deploy/deployment.yaml
```

Testing it Out

You can create a test PVC to ensure that your GlusterFS and provisioner are working correctly together with Kubernetes:

deploy-glfs.sh

```
$ kubectl create -f deploy/pvc.yaml
persistentvolumeclaim/gluster-simple-claim created
```

You should see that you PVC is created with an initial state of Pending and no PersistentVolume has been provisioned for it:

deploy-glfs.sh

```
$ kubectl get pvc,pv
NAME                                        STATUS    VOLUME    CAPACITY   ACCESS MODES   STORAGECLASS
AGE
persistentvolumeclaim/gluster-simple-claim   Pending                                       glusterfs-simple   2s
```

After a few seconds, a volume will be provisioned for your PVC:

deploy-glfs.sh

```
$ kubectl get pvc,pv
NAME                                        STATUS    VOLUME                                     CAPACITY
ACCESS MODES   STORAGECLASS       AGE
persistentvolumeclaim/gluster-simple-claim   Bound     pvc-e519c597-a195-11e8-82d6-fa163e59d79f   1Gi
RWX            glusterfs-simple   5s

NAME                                                    CAPACITY   ACCESS MODES   RECLAIM POLICY
STATUS    CLAIM                             STORAGECLASS       REASON    AGE
persistentvolume/pvc-e519c597-a195-11e8-82d6-fa163e59d79f   1Gi        RWX            Delete
Bound     default/gluster-simple-claim   glusterfs-simple             2s
```

You can exec into the glusterfs pods to verify that a gluster volume was create for your PVC, and check the provisioner pod logs to see how it all happened under the hood:

deploy-glfs.sh

```
$ kubectl get pods
NAME                                          READY     STATUS    RESTARTS   AGE
glusterfs-simple-provisioner-86c6d8c8cd-75bk4   1/1       Running   0          5h
glusterfs-t44m5                               1/1       Running   0          5h
glusterfs-v64wn                               1/1       Running   0          5h


$ kubectl exec -it glusterfs-t44m5 -- gluster volume list
pvc-e519c597-a195-11e8-82d6-fa163e59d79f


$ kubectl logs -f glusterfs-simple-provisioner-86c6d8c8cd-75bk4
I0816 15:50:58.969822       1 main.go:47] Provisioner gluster.org/glusterfs-simple specified
I0816 15:50:58.969896       1 main.go:56] Building kube configs for running in cluster...
I0816 15:50:58.988158       1 provision.go:45] Creating NewGlusterfsProvisioner.
I0816 15:50:58.988635       1 leaderelection.go:185] attempting to acquire leader lease  kube-system/gluster.
org-glusterfs-simple...
I0816 15:50:59.000100       1 leaderelection.go:194] successfully acquired lease kube-system/gluster.org-
glusterfs-simple
I0816 15:50:59.000155       1 event.go:221] Event(v1.ObjectReference{Kind:"Endpoints", Namespace:"kube-system",
Name:"gluster.org-glusterfs-simple", UID:"2b4eef67-a16c-11e8-82d6-fa163e59d79f", APIVersion:"v1",
ResourceVersion:"1165", FieldPath:""}): type: 'Normal' reason: 'LeaderElection' glusterfs-simple-provisioner-
86c6d8c8cd-75bk4_2b4e2946-a16c-11e8-ab87-0a580af40102 became leader
I0816 15:50:59.000203       1 controller.go:596] Starting provisioner controller gluster.org/glusterfs-
simple_glusterfs-simple-provisioner-86c6d8c8cd-75bk4_2b4e2946-a16c-11e8-ab87-0a580af40102!
I0816 15:50:59.100536       1 controller.go:645] Started provisioner controller gluster.org/glusterfs-
simple_glusterfs-simple-provisioner-86c6d8c8cd-75bk4_2b4e2946-a16c-11e8-ab87-0a580af40102!

    ...    ...    ...    ...    ...    ...    ...    ...    ...    ...    ...    ...    ...    ...    ...
...    ...    ...    ...    ...    ...    ...    ...    ...


I0816 20:49:40.074522       1 provision.go:183] mkdir -p 192.168.0.9:/tmp/default/gluster-simple-claim-pvc-
e519c597-a195-11e8-82d6-fa163e59d79f
I0816 20:49:40.074750       1 event.go:221] Event(v1.ObjectReference{Kind:"PersistentVolumeClaim", Namespace:"
default", Name:"gluster-simple-claim", UID:"e519c597-a195-11e8-82d6-fa163e59d79f", APIVersion:"v1",
ResourceVersion:"37040", FieldPath:""}): type: 'Normal' reason: 'Provisioning' External provisioner is
provisioning volume for claim "default/gluster-simple-claim"
I0816 20:49:40.080309       1 exec.go:108] Pod selecterd: default/glusterfs-t44m5
I0816 20:49:40.182105       1 exec.go:81] Result:
I0816 20:49:40.182132       1 exec.go:82] Result:
I0816 20:49:40.277435       1 exec.go:81] Result:
I0816 20:49:40.277462       1 exec.go:82] Result:
I0816 20:49:40.375121       1 exec.go:81] Result:
I0816 20:49:40.375158       1 exec.go:82] Result:
I0816 20:49:40.375171       1 provision.go:183] mkdir -p 192.168.0.4:/tmp/default/gluster-simple-claim-pvc-
e519c597-a195-11e8-82d6-fa163e59d79f
I0816 20:49:40.378560       1 exec.go:108] Pod selecterd: default/glusterfs-v64wn
I0816 20:49:40.501549       1 exec.go:81] Result:
```

```
I0816 20:49:40.501579       1 exec.go:82] Result:
I0816 20:49:40.630585       1 exec.go:81] Result:
I0816 20:49:40.630608       1 exec.go:82] Result:
I0816 20:49:40.737097       1 exec.go:81] Result:
I0816 20:49:40.737193       1 exec.go:82] Result:
I0816 20:49:40.741076       1 exec.go:108] Pod selecterd: default/glusterfs-t44m5
I0816 20:49:41.072344       1 exec.go:81] Result: volume create: pvc-e519c597-a195-11e8-82d6-fa163e59d79f:
success: please start the volume to access data
I0816 20:49:41.072370       1 exec.go:82] Result:
I0816 20:49:43.536546       1 exec.go:81] Result: volume start: pvc-e519c597-a195-11e8-82d6-fa163e59d79f:
success
I0816 20:49:43.536585       1 exec.go:82] Result:
I0816 20:49:43.559744       1 controller.go:1043] volume "pvc-e519c597-a195-11e8-82d6-fa163e59d79f" for claim
"default/gluster-simple-claim" created
I0816 20:49:43.568855       1 controller.go:1060] volume "pvc-e519c597-a195-11e8-82d6-fa163e59d79f" for claim
"default/gluster-simple-claim" saved
I0816 20:49:43.568887       1 controller.go:1096] volume "pvc-e519c597-a195-11e8-82d6-fa163e59d79f" provisioned
for claim "default/gluster-simple-claim"
I0816 20:49:43.569213       1 event.go:221] Event(v1.ObjectReference{Kind:"PersistentVolumeClaim", Namespace:"
default", Name:"gluster-simple-claim", UID:"e519c597-a195-11e8-82d6-fa163e59d79f", APIVersion:"v1",
ResourceVersion:"37040", FieldPath:""}): type: 'Normal' reason: 'ProvisioningSucceeded' Successfully
provisioned volume pvc-e519c597-a195-11e8-82d6-fa163e59d79f
```