GlusterFS cluster with Kubernetes

Jaime Garza · Follow

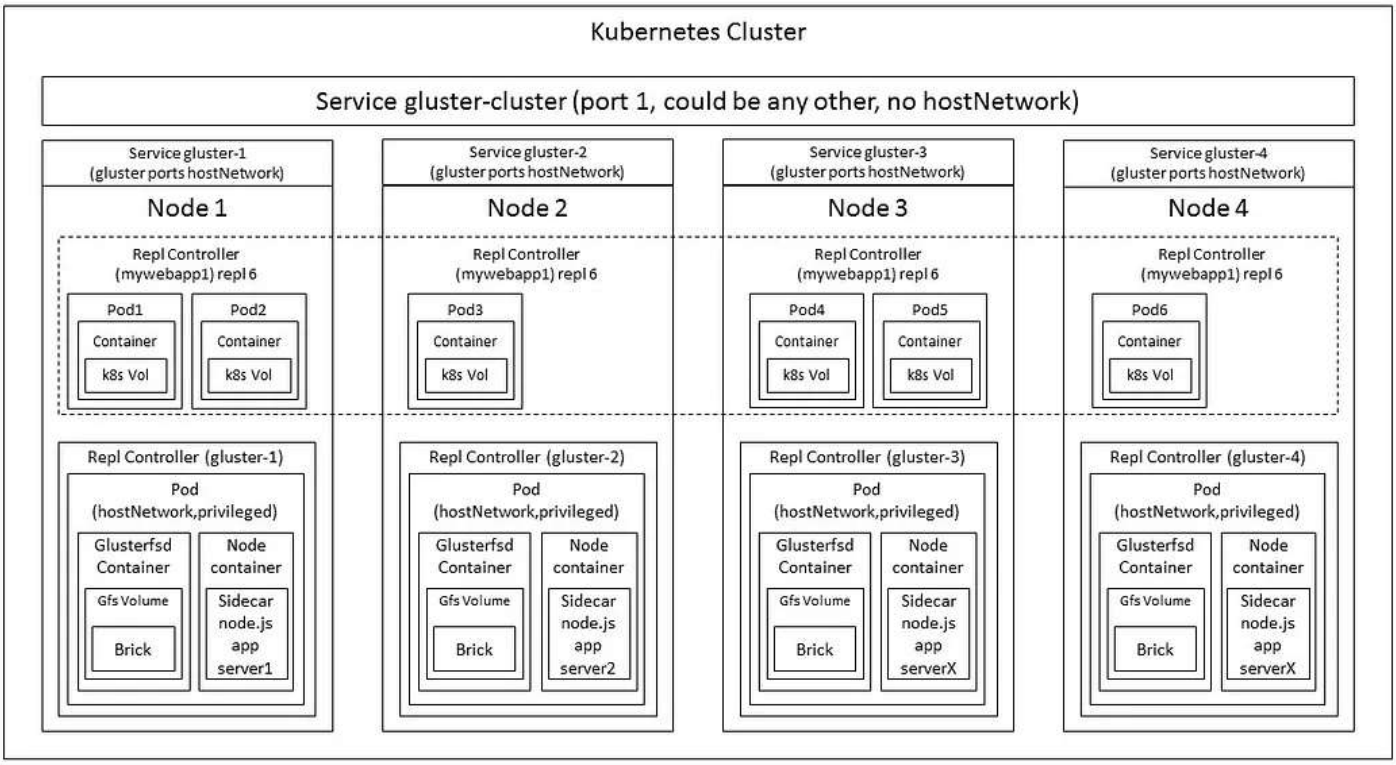
6 min read · May 22, 2016

This is a POC trying to run both a cluster of GlusterFS nodes on top of kubernetes as containers, with the sidecar pattern (also inspired in a MongoDB sidecar article) and able to be mounted on a k8s pod with the kubernetes GlusterFS volume plugin.

This sidecar has some limitations on the GlusterFS functionality, it was designed to have just **one volume** per cluster, have a factor replication of minimum **2**, also as per the GlusterFS dependency on important ports as the **111** for **udp** and **tcp** mapping, when creating k8s replication controllers and services we need to set **hostNetwork** to **true**, so that means we are limited to have a number of GlusterFS nodes **no bigger** than the number of **physical nodes** available on the kubernetes cluster.

(Yes I know… this breaks the principles of kubernetes and containers, but that's the only way I managed to get it working)

So an implementation of this sidecar looks similar to this:



kubernetes poc diagram of gluster cluster

The podtemplate spec works with 2 docker images I created that are not official, the one of the sidecar, and the other of a custom debian glusterfs container. You can change this to your own custom image or use the non-official.

Gluster daemon container

https://hub.docker.com/r/neshte/gluster-debian-jessie/

https://github.com/neshte/gluster-debian-jessie

Sidecar

https://hub.docker.com/r/neshte/gluster-k8s-sidecar/

https://github.com/neshte/gluster-k8s-sidecar

So what are the steps to get this? (Note: this was just tested with Google Cloud Platform, it could be adapted to other vendors like AWS, etc.)

Step 0:

Assuming you are playing over Google Cloud Platform. You need to make sure to have well configured:

- Your **gcloud** command with your credentials configured and everything

```
$ gcloud init
```

- As well as your **kubectl** command pointing to your kubernetes cluster with the credentials and everything

```
$ gcloud config set compute/zone us-central1-b
$ gcloud components install kubectl
$ gcloud container clusters get-credentials NAME
```

Step 1:

Create leader

```
$ git clone https://github.com/neshte/gluster-k8s-sidecar.git
$ cd gluster-k8s-sidecar/examples
```

```
$ make add-glusternode
```

There are some variables you can modify to this:

- **DISK_SIZE**=1000
  The size of the disk in GB you are going to create on your Google Cloud Platform
- **ZONE**=us-central1-b
  If your cluster is on a different zone than your default, set it

The other variables you can change are on the Replication Controller template

```
$ ls
emptydir_pod_examples                       Makefile
gluster-controller-flocker-template.yaml    podtest.json
gluster-controller-template.yaml            README.md
gluster-service-template.yaml               svctest.json
```

- Selector labels (according to your needs)

```
kind: ReplicationController
apiVersion: v1
...
  selector:
    name: gluster-node-<num>
    role: gluster
    environment: test
  template:
    spec:
...
        - name: gluster-sidecar
          image: neshte/gluster-k8s-sidecar
          env:
```

```
            - name: GLUSTER_SIDECAR_POD_LABELS
              value: "role=gluster,environment=test"
...
    metadata:
      labels:
        name: gluster-node-<num>
        role: gluster
        environment: test
```

And you can override the following defaults in the same “env” section:

- **GLUSTER_SIDECAR_SLEEP_SECONDS**
  Default: 5
  This is how long to sleep between work cycles. Depending on your cluster power or your real-time reconfiguration needs, or the amount of logs you want to accumulate, you may want to change this.
- **GLUSTER_SIDECAR_UNHEALTHY_SECONDS**
  Default: 15
  This is how many seconds the sidecar will sleep its workloop if something is not working as expected and there is some kind of failure.
- **GLUSTER_SIDECAR_CLUSTER_NAME**
  Default: glusterfs-cluster
  This is the meta.name of the kubernetes endpoints.
- **GLUSTER_SIDECAR_CLUSTER_PORT**
  Default: 1
  This is the cluster port for kubernetes gluster volume mounting.
- **GLUSTER_SIDECAR_VOLUME_NAME**
  Default: data
  This is the gluster volume name.

- **GLUSTER_SIDECAR_BRICK_NAME**
  Default: data
  This is the gluster brick name.

- **GLUSTER_SIDECAR_REPLICATION**
  Default: 2
  This is the replication factor for files on gluster cluster. Not tested with a number lower than 2.

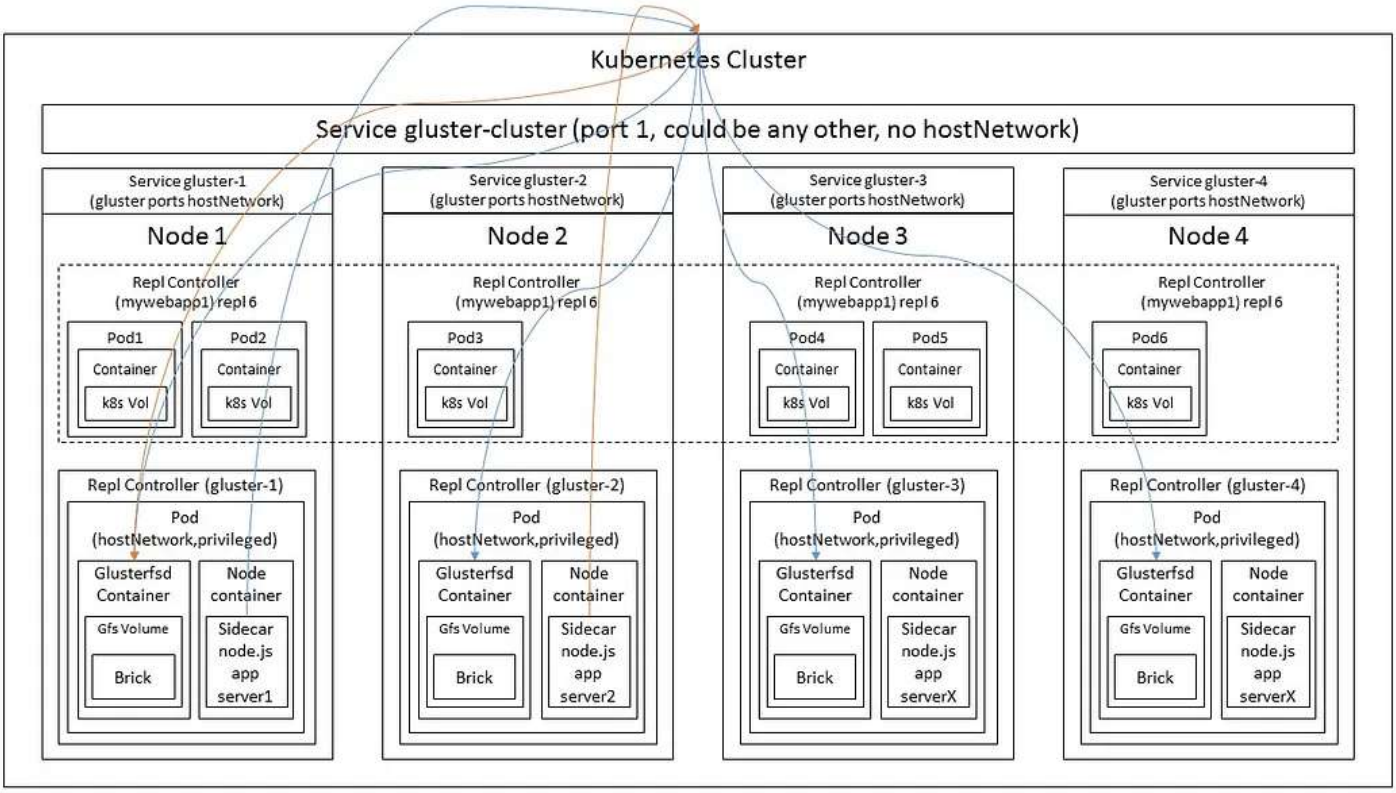
- **GLUSTER_SIDECAR_K8SNAMESPACE**
  Default: default
  This is the namespace in which this sidecar will run.

Step 2:

Repeat the process to have a total number of replication controllers in multiples of your replication factor (default 2). For example, to have 4 gluster nodes repeat the step 1, 3 more times:

```
$ make add-glusternode
$ make add-glusternode
$ make add-glusternode
```

Logic of sidecars



orange are connections from server2 sidecar, blue are connections from server1 sidecar

When no pod exists, the first one (**server1**) queries kubernetes pods with the corresponding labels and based on that decides that it will be the leader. All the debug logging happens on **server1.**

When a pod exists, it checks to see if is the second pod, if it is the second it will be **server2.**

When **server1** finds that there is a new **server2**, it probes the **server2** to create the **trusted storage pool.** When the **trusted storage pool** is first created, also a **volume** is created and also a **brick** in each of **server1** and **server2** is created.

When more pods are added (**serverX**'s) each of them finds out there is already a **server1** and **server2**, so they should just wait to be invited by them to the cluster.

When **server1** finds out a third, fourth, etc. pod exists (**serverX**) it probes that **serverX** to invite it to the cluster. Then **server1** creates on those **serverX**'s their corresponding bricks.

The number of **pods/rc/glusternodes** needs to be a multiple of the replication factor. So if **server1** finds a **serverX** but it needs to find 2 **serverX** for it to reconfigure the cluster to auto-expand, then it will wait until there are enough **serverX**'s for the expansion to happen.

If **server1** dies, **server2** will now be **server1** and the oldest created **serverX** will be the new **server2**.

If a **serverX** dies and it was already part of the cluster, then **server1** will wait until there is a new **serverX** waiting to be invited to the cluster, and reconfigure it by copying the data from a server with the replicas of the data that was lost.

(Note: this has not been tested very roughly, if you have a pull-request or issue, it will be happily accepted)

Step 3:

How to attach this new cluster as a folder in your application for persistent distributed scalable storage.

Basically from this step, you now just need to implement what is said in the official Kubernetes GlusterFS volume plugin documentation.

kubernetes/kubernetes

kubernetes - Production-Grade Container Scheduling and Management

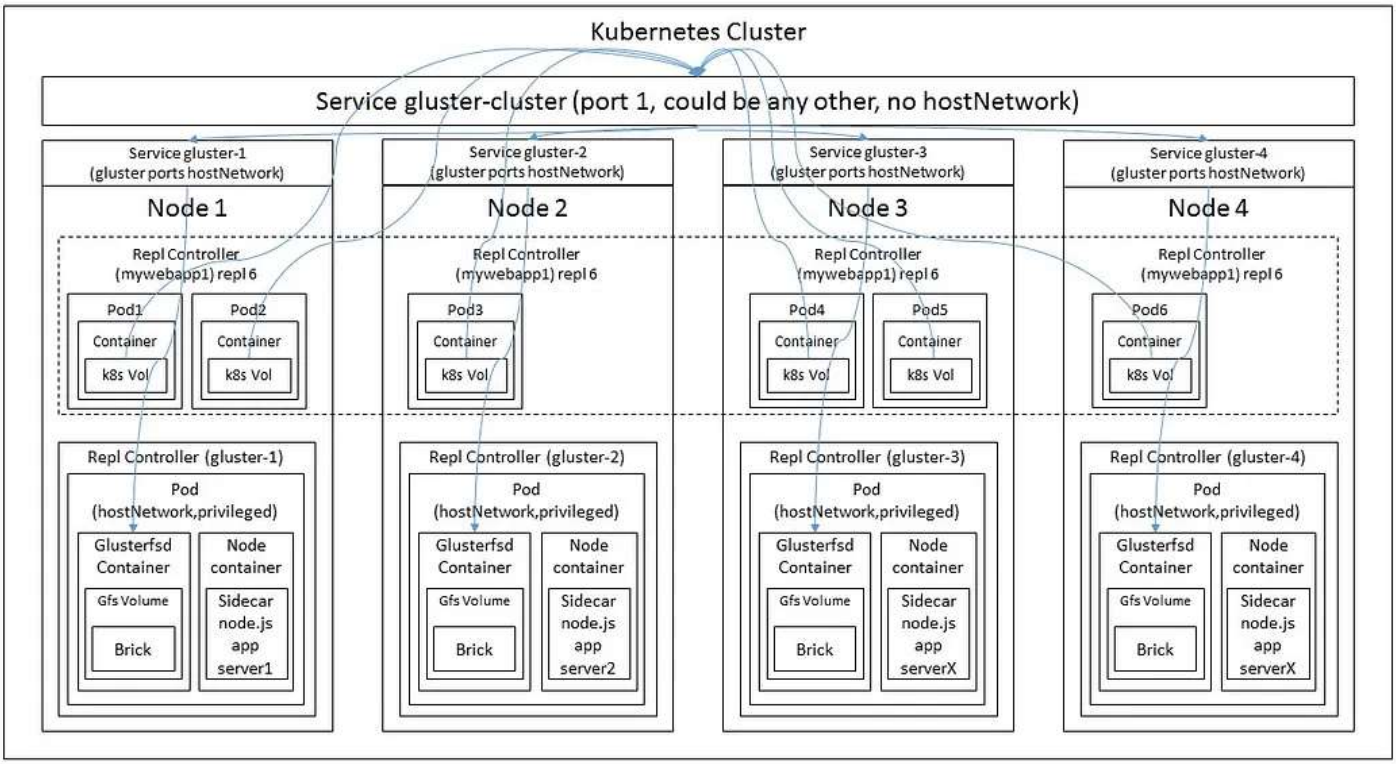
github.com

In your app's replication controller's pod template or in your pod definition, just make sure to have the glusterfs cluster you have just created, mounted as a volume.

```
{
    "apiVersion": "v1",
    "kind": "Pod",
    "metadata": {
        "name": "mywebapp1"
    },
    "spec": {
        "containers": [
            {
                "name": "glusterfs",
                "image": "example/mywebapp1",
                "volumeMounts": [
                    {
                        "mountPath": "/mnt/glusterfs",
                        "name": "glusterfsvol"
                    }
                ]
            }
        ],
        "volumes": [
            {
                "name": "glusterfsvol",
                "glusterfs": {
                    "endpoints": "glusterfs-cluster",
                    "path": "data"
                }
            }
```

```
            ]
        }
    }
```

And what happens is something like this:



your app has a folder which is a mount of a gluster, which is available thanks to the gluster plugin and the gluster-cluster service you created, which then routes to the different gluster-N services, and those services finally talk to the gluster daemon of each pod in each rc, which has the volume which has the brick which has your data!

Important:

This is a POC, is not production ready, we are not sure of the implications in performance neither on security, and I'm open to hear feedback about this approach.

If you don't like this approach there are others doing similar things, maybe you prefer one of those. If you prefer other option different to this one, or if

you simply have a different way to solve the distributed scalable storage on Kubernetes with something different to Gluster, I will appreciate if you send me a comment with your stuff. Here are the links:

wattsteve/glusterfs-kubernetes
jsafrane/glusterfs-mounter
http://blog.xebia.com/persistence-with-docker-containers-team-1-glusterfs-2/
https://huaminchen.wordpress.com/2015/03/31/use-glusterfs-as-persistent-storage-in-kubernetes/
http://events.linuxfoundation.org/sites/events/files/slides/Lessons%20Learned%20Containerizing%20GlusterFS%20and%20Ceph%20with%20Docker%20and%20Kubernetes.pdf
http://website-humblec.rhcloud.com/gluster_containers_in_kubernetes_cluster/
http://website-humblec.rhcloud.com/glusterfs-containers-docker-kubernetes-openshift/

Kubernetes Docker Glusterfs