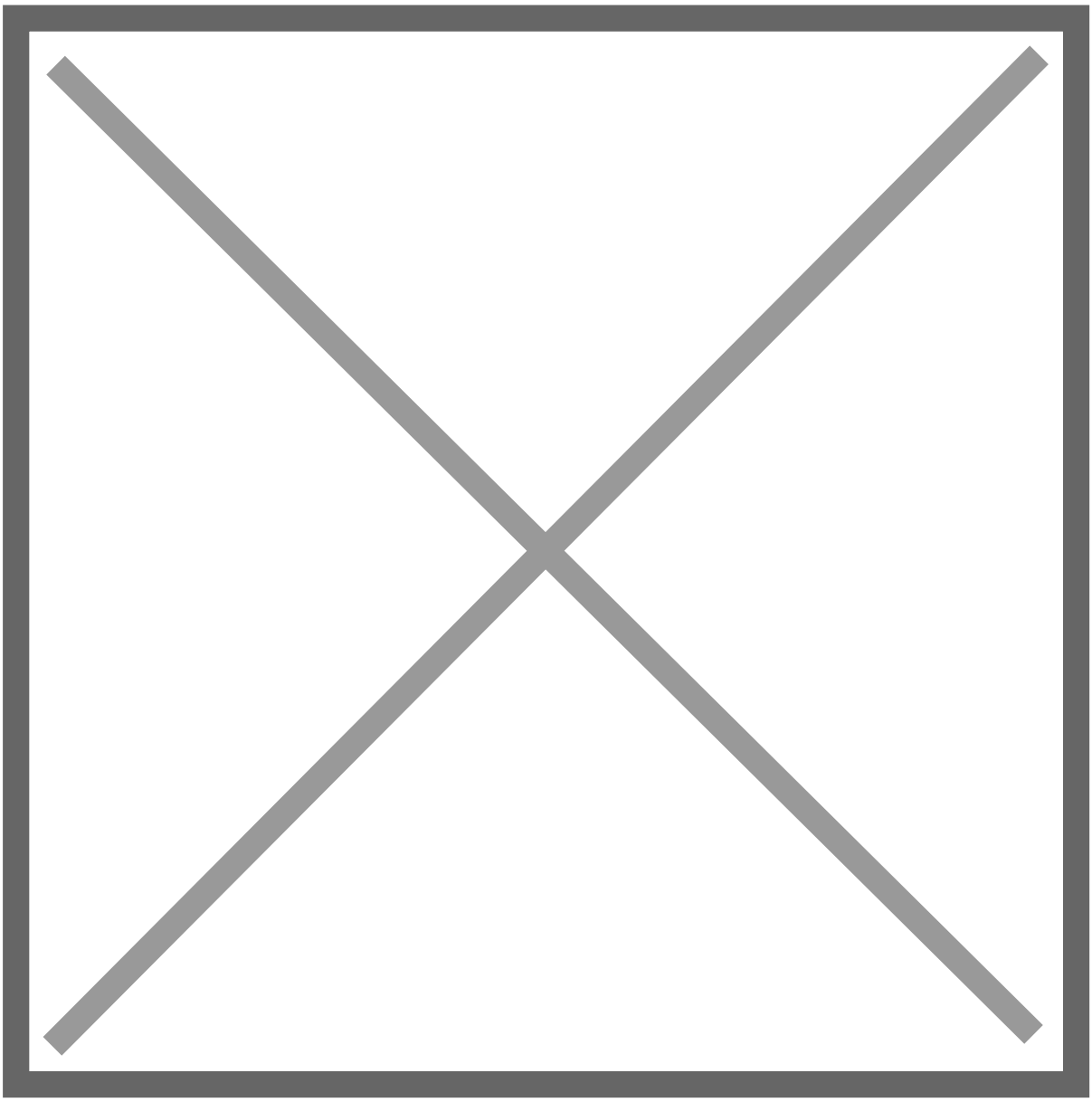
Setup Highly Available applications with Docker Swarm and Gluster

Link: https://medium.com/running-a-software-factory/setup-3-node-high-availability-cluster-with-glusterfs-and-docker-swarm-b4ff80c6b5c3

Published in Oct 28, 2018



Docker Swarm cluster with shared glusterFS replicated volume for HA

A good design pattern for highly available applications is to deploy the application as a container on a Docker Swarm cluster with persistent storage provided by GlusterFS. GlusterFS is a fast shared filesystem that can keep the container volume in sync between multiple VMs running the Docker Swarm cluster. This pattern ensures high availability for your containerised application. In the event a VM dies, Docker Swarm will spin up the container on another VM. GlusterFS will ensure the container has access to the same data when it comes up.

In this tutorial, we'll look at setting up GlusterFS on 3 VMs and create a replicated volume with a replication factor of 3. Later we'll install Docker Swarm over these three VMs. Goal is to use GlusterFS to provide persistent storage to your application container, and docker swarm for high availability.

1. Plan and setup the infrastructure

For the setup, first we'll need three Ubuntu Gluster VMs, each with 2 disks attached. We'll use the first disk to run the OS, and the second as the GlusterFS replicated volume. Create three VMs with two disks. In my case, my VMs had the root volume on `/dev/vda` and the second disk on `/dev/vdc`. Create three VMs and let's assume the private IPs of these VMs are `192.168.2.100, 192.168.2.101, 192.168.2.102`, and their hostnames are `gluster1, gluster2, gluster3`.

Note: All commands are being executed as `root` user (hence the # at the beginning)

```
# lsblk
NAME   MAJ:MIN RM SIZE RO TYPE MOUNTPOINT
vda    253:0    0  30G  0 disk
└─vda1 253:1    0  30G  0 part /
vdb    253:16   0  64M  0 disk
vdc    253:32   0  10G  0 disk
```

Update the /etc/hosts files on each VM to reflect the private IPs of each VM. This is important for GlusterFS, and you may encounter bugs or issues if you give private IPs directly to Gluster volumes. After editing the files should look like:

```
(gluster1)# cat /etc/hosts
127.0.0.1       localhost
192.168.2.100   gluster1
192.168.2.101   gluster2
192.168.2.102   gluster3(gluster2)# cat /etc/hosts
127.0.0.1       localhost
192.168.2.100   gluster1
192.168.2.101   gluster2
192.168.2.102   gluster3(gluster3)# cat /etc/hosts
127.0.0.1       localhost
192.168.2.100   gluster1
192.168.2.101   gluster2
192.168.2.102   gluster3
```

Format the disks with xfs filesystem on each VM in case you haven't already. You can also use ext4 if you prefer.

```
# mkfs.xfs /dev/vdc
```

2. Create directories for GlusterFS storage

Setup the glusterFS directories where the gluster “bricks” will reside. Better to name them differently so it’s easy to identify on which node the replicated volumes reside. Also add an entry to your /etc/fstab file on each VM so that our brick gets mounted when the operating system boots or restarts.

```
(gluster1)# mkdir -p /gluster/bricks/1
(gluster1)# echo '/dev/vdc /gluster/bricks/1 xfs defaults 0 0' >> /etc/fstab
(gluster1)# mount -a
(gluster1)# mkdir /gluster/bricks/1/brick(gluster2)# mkdir -p /gluster/bricks/2
(gluster2)# echo '/dev/vdc /gluster/bricks/2 xfs defaults 0 0' >> /etc/fstab
(gluster2)# mount -a
(gluster2)# mkdir /gluster/bricks/2/brick(gluster3)# mkdir -p /gluster/bricks/3
(gluster3)# echo '/dev/vdc /gluster/bricks/3 xfs defaults 0 0' >> /etc/fstab
(gluster3)# mount -a
(gluster3)# mkdir /gluster/bricks/3/brick
```

3. Install GlusterFS

Install GlusterFS on all VMs by executing following commands on each VM:

```
# apt-get -y update && apt-get -y upgrade
# apt-get install -y software-properties-common
# add-apt-repository ppa:gluster/glusterfs-6 && apt-get update # Use the latest glusterFS versio
# apt-get install -y glusterfs-server
# systemctl enable glusterd # automatically start glusterfs on boot
# systemctl start glusterd # start glusterfs right now
# systemctl status glusterd # Should show status active
```

4. Peer with other Gluster VMs

Now peer with other nodes from gluster1:

```
(gluster1)# gluster peer probe gluster2
peer probe: success.
(gluster1)# gluster peer probe gluster3
peer probe: success.
(gluster1)# gluster peer status
Number of Peers: 2Hostname: gluster2
Uuid: 60861905-6adc-4841-8f82-216c661f9fe2
State: Peer in Cluster (Connected)Hostname: gluster3
Uuid: 572fed90-61de-40dd-97a6-4255ed8744ce
State: Peer in Cluster (Connected)
```

5. Setup the Gluster “replicated volume”

GlusterFS has multiple volume types. For our HA architecture, we want to setup a “replicated” volume that stores the files created on each of the 3 VMs and hence the file is available to any app or container running on these VMs. Create the replicated volume named “gfs” with 3 replicas:

```
(gluster1)# gluster volume create gfs \
replica 3 \
gluster1:/gluster/bricks/1/brick \
gluster2:/gluster/bricks/2/brick \
gluster3:/gluster/bricks/3/brickvolume create: gfs: success: please start the volume to access d
(gluster1)# gluster volume status gfsStatus of volume: gfs
Gluster process                             TCP Port  RDMA Port  Online  Pid
------------------------------------------------------------------------------
Brick gluster1:/gluster/bricks/1/brick      49152     0          Y       4619
Brick gluster2:/gluster/bricks/2/brick      49152     0          Y       4504
Brick gluster3:/gluster/bricks/3/brick      49152     0          Y       4306
Self-heal Daemon on localhost               N/A       N/A        Y       4641
Self-heal Daemon on gluster2                N/A       N/A        Y       4526
Self-heal Daemon on gluster3                N/A       N/A        Y       4328Task Status of Volu
------------------------------------------------------------------------------
There are no active volume tasks(gluster1)# gluster volume info gfsVolume Name: gfs
Type: Replicate
Volume ID: 703e46cb-a637-4620-adfa-6b292a15e0d5
Status: Started
Snapshot Count: 0
Number of Bricks: 1 x 3 = 3
Transport-type: tcp
Bricks:
Brick1: gluster1:/gluster/bricks/1/brick
Brick2: gluster2:/gluster/bricks/2/brick
Brick3: gluster3:/gluster/bricks/3/brick
Options Reconfigured:
```

```
transport.address-family: inet
nfs.disable: on
performance.client-io-threads: off
```

6. Setup security and authentication for this volume

GlusterFS will allow any clients to connect to volumes by default. However, you will need to authorize the three infra VMs running GlusterFS to connect to the GlusterFS Volumes on each node. You can do it by authorizing the private IPs of each VM to connect to the volume. This will allow replication to happen. Execute:

```
(gluster1)# gluster volume set gfs auth.allow 192.168.2.100,192.168.2.101,192.168.2.102
```

7. Mount the glusterFS volume where applications can access the files

We'll mount the volume onto `/mnt` on each VM, and also append it to our /etc/fstab file so that it mounts on boot:

```
(gluster1)# echo 'localhost:/gfs /mnt glusterfs defaults,_netdev,backupvolfile-server=localhost
(gluster1)# mount.glusterfs localhost:/gfs /mnt(gluster2)# echo 'localhost:/gfs /mnt glusterfs d
(gluster2)# mount.glusterfs localhost:/gfs /mnt(gluster3)# echo 'localhost:/gfs /mnt glusterfs d
(gluster3)# mount.glusterfs localhost:/gfs /mnt
```

8. Verify

Verify mounted glusterfs volume:

```
# df -Th
Filesystem     Type            Size  Used Avail Use% Mounted on
udev           devtmpfs        7.9G     0  7.9G   0% /dev
tmpfs          tmpfs           1.6G   17M  1.6G   2% /run
/dev/vda1      ext4             30G  2.1G   27G   8% /
tmpfs          tmpfs           7.9G   12K  7.9G   1% /dev/shm
tmpfs          tmpfs           5.0M     0  5.0M   0% /run/lock
tmpfs          tmpfs           7.9G     0  7.9G   0% /sys/fs/cgroup
tmpfs          tmpfs           1.6G     0  1.6G   0% /run/user/1001
/dev/vdb       xfs              10G   33M   10G   1% /gluster/bricks/1
localhost:/gfs fuse.glusterfs   10G  135M   10G   2% /mnt
```

The total space available on the volume comes up as 10G even though we have 3 disks of 10G each connected to GlusterFS. This is due to our replication factor of 3. Total volume size is 30G, but with a replication factor or 3 for each file only 10G is available to us.

Test GlusterFS replication:

```
(gluster1)# echo "Hello World!" | sudo tee /mnt/test.txt(gluster2)# cat /mnt/test.txt
Hello World!(gluster3)# cat /mnt/test.txt
Hello World!
```

Part 2: Setup Docker Swarm

Now let’s setup the Docker Swarm cluster with the gluster VMs (gluster1/2/3) as the workers, and a new VM (swarm-manger) as the Swarm manager. We’ll use our gluster replicated volume to achieve High Availability for our stateful containerized application. We’ll test with Wordpress.

All commands executed as root.

1. Setup Docker community edition on all VMs

Install docker-ce on all four VMs (swarm-manager, gluster1/2/3) using the instructions given here: https://docs.docker.com/install/linux/docker-ce/ubuntu/ (I feel it’s redundant to repeat the standard instructions).

However, after the installation, please do verify if Docker is installed properly by running following command on all VMs:

```
# docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
1b930d010525: Pull complete
Digest: sha256:92695bc579f31df7a63da6922075d0666e565ceccad16b59c3374d2cf4e8e50e
Status: Downloaded newer image for hello-world:latestHello from Docker!
This message shows that your installation appears to be working correctly.
```

2. Initialize Docker swarm from the swarm-manager

We'll use the swarm-manager's private IP as the "advertised address".

```
swarm-manager:~# docker swarm init --advertise-addr 192.168.2.99
Swarm initialized: current node (sz42o1yjz08t3x98aj82z33pe) is now a manager.To add a worker to
```

3. Add the three gluster VMs as swarm workers

```
gluster1:~# docker swarm join --token SWMTKN-1-3gi2wi4o22nyiqij3io055na7wt0201oamaegykllea0t5vi5
This node joined a swarm as a worker.gluster2:~# docker swarm join --token SWMTKN-1-3gi2wi4o22ny
This node joined a swarm as a worker.gluster3:~# docker swarm join --token SWMTKN-1-3gi2wi4o22ny
This node joined a swarm as a worker.swarm-manager:~# docker node ls
ID                            HOSTNAME            STATUS              AVAILABILITY        MANAGE
qjmuz0n8n770ryougk2tsb37x     gluster1            Ready               Active
kcwsavrtzhvy038357p51lwl2     gluster2            Ready               Active
ifnzgpk25p27y19vslee4v74x     gluster3            Ready               Active
sz42o1yjz08t3x98aj82z33pe *   swarm-manager       Ready               Active              Leader
```

Part 3: Test the High Availability Setup

We'll use docker stack to setup a single container Wordpress backed by a single container of MySQL, and then test if this setup is resilient to VM failure.

1. Create replicated directories for wordpress and mysql in glusterFS

```
gluster1:~# mkdir /mnt/wp-content
gluster1:~# mkdir /mnt/mysql
```

2. Create the wordpress-stack.yml file

This stack file exposes wordpress on port 8080 on all swarm nodes, even the swarm-manager node. It mounts the directories created for wp-content and mysql as volumes on the containers.

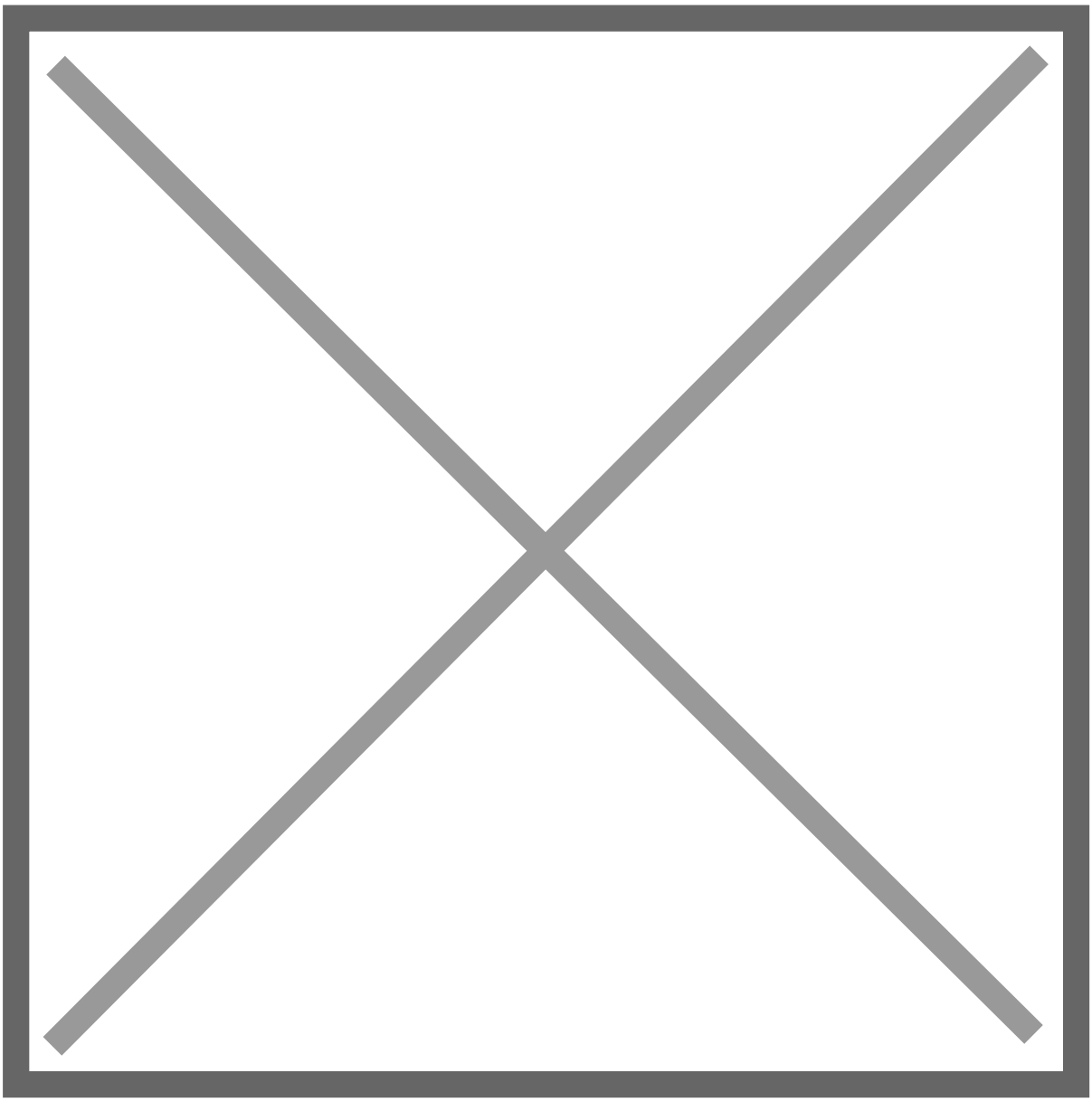
```
swarm-manager:~# cat wordpress-stack.yml
# wordpress-stack.yml
version: '3.1'services:  wordpress:
    image: wordpress
    restart: always
    ports:
      - 8080:80
    environment:
      WORDPRESS_DB_HOST: db
      WORDPRESS_DB_USER: exampleuser
```

```
      WORDPRESS_DB_PASSWORD: examplepass
      WORDPRESS_DB_NAME: exampledb
    volumes:
      - "/mnt/wp-content:/var/www/html/wp-content"
    deploy:
      placement:
        constraints: [node.role == worker]  db:
    image: mysql:5.7
    restart: always
    environment:
      MYSQL_DATABASE: exampledb
      MYSQL_USER: exampleuser
      MYSQL_PASSWORD: examplepass
      MYSQL_RANDOM_ROOT_PASSWORD: '1'
    volumes:
      - "/mnt/mysql:/var/lib/mysql"
    deploy:
      placement:
        constraints: [node.role == worker]
```

3. Use docker stack to deploy Wordpress and MySQL

```
swarm-manager:~# docker stack deploy -c wordpress-stack.yml wordpress
Ignoring unsupported options: restartCreating network wordpress_default
Creating service wordpress_db
Creating service wordpress_wordpressswarm-manager:~# docker stack ps wordpress
ID                  NAME                    IMAGE               NODE                DESIRED STAT
x5vvrt6ohko2        wordpress_db.1          mysql:5.7           gluster2            Running
idree9r7qlxb        wordpress_wordpress.1   wordpress:latest    gluster1            Running
```

Check if Wordpress is up by entering http://<any-worker-external-ip>:8080/ in the browser.



Note: 10.147.106.32 was one of my gluster worker VM’s (gluster3) external IP

Go through the install process, choose an admin username and password, and create your first post.

4. Test High Availability by shutting down a VM

Check on which VM the Wordpress and MySQL containers are running. We’ll shutdown each VM to understand whether HA is working properly. In my case, the Wordpress container was running on gluster1 and MySQL was running on gluster2.

```
swarm-manager:~# docker stack ps wordpress
ID                  NAME                    IMAGE               NODE                DESIRED STAT
x5vvrt6ohko2        wordpress_db.1          mysql:5.7           gluster2            Running
idree9r7qlxb        wordpress_wordpress.1   wordpress:latest    gluster1            Running
```

Shutdown gluster1 and check what happens. You'll find that docker swarm starts a new container on a new worker VM. The website will continue to work, your data will still be stored, but you'll have to login again as the session data is lost with the previous container.

```
swarm-manager:~# docker stack ps wordpress
ID                  NAME                    IMAGE               NODE                DESIRED STAT
u8s93kowj2mx        wordpress_wordpress.1   wordpress:latest    gluster3            Running
x5vvrt6ohko2        wordpress_db.1          mysql:5.7           gluster2            Running
idree9r7qlxb        wordpress_wordpress.1   wordpress:latest    gluster1            Shutdown
```

Start the gluster1 VM again and let's repeat the HA test with MySQL host gluster2. Shutdown gluster2 which was running the MySQL container. After shutdown, you'll find docker swarm has scheduled MySQL on another worker VM.

```
swarm-manager:~# docker stack ps wordpress
ID                  NAME                    IMAGE               NODE                DESIRED STAT
px90rs5q22ei        wordpress_db.1          mysql:5.7           gluster1            Running
u8s93kowj2mx        wordpress_wordpress.1   wordpress:latest    gluster3            Running
x5vvrt6ohko2        wordpress_db.1          mysql:5.7           gluster2            Shutdown
idree9r7qlxb        wordpress_wordpress.1   wordpress:latest    gluster1            Shutdown
```

The website will continue to work without any data loss as the MySQL container would have found the replicated volume under the same path (/mnt/mysql).

Add the three worker VM IPs with port behind a Load Balancer (like AWS ELB) and *voilà*, A Highly Available stateful deployment on Docker Swarm using GlusterFS.

Revision #1
Created 2024-07-08 10:24:33 UTC by Administrador
Updated 2024-07-08 10:27:53 UTC by Administrador