GlusterFS

Aplicativo de mapeamento de sistemas de arquivos/discos

- Configurações GlusterFS
  - Tutorial: Create a Docker Swarm with Persistent Storage Using GlusterFS
  - Tutorial: Deploy a Highly Availability GlusterFS Storage Cluster
  - Setup a scalable high availability GlusterFS network filesystem with Docker Swarm on Ubuntu Server 20.04 LTS
  - Setup Highly Available applications with Docker Swarm and Gluster
  - Configuração GlusterFS Docker
  - Configuração GlusterFS Docker2

Configurações GlusterFS

Configurações GlusterFS

Tutorial: Create a Docker Swarm with Persistent Storage Using GlusterFS

Link: https://thenewstack.io/tutorial-create-a-docker-swarm-with-persistent-storage-using-glusterfs/

How to use GlusterFS to provide persistent storage for a Docker Swarm.

Apr 2nd, 2020 12:00pm by Jack Wallen
Featued image for: Tutorial: Create a Docker Swarm with Persistent Storage Using GlusterFS

Feature image: The GlusterFS Ant, Gluster.

Unleashing a Docker Swarm orchestrator is a great (and relatively easy) way to deploy a container cluster. Yes, you could go with Kubernetes for more management features, but when you need the bare bones of a simple container cluster, Docker Swarm is a pretty good way to go.

The one thing you might find yourself needing is persistent storage for your cluster. What is persistent storage? I'm glad you asked. To put it simply, persistent storage is any type of data storage device that retains data, even after power to the device is cut off. With regards to a container, persistent storage is storage that remains, even if the container isn't running. In other words, persistent storage is found on the hosting server, so when the container is spun down, the data within the storage is still accessible. Or, if the container is a part of the swarm, that persistent storage can be shared between nodes.

To any container developer, persistent storage is often a must-use tool. With some container technology, persistent storage can be done quite simply. Although with Docker you can use volumes, the problem with that feature is that it is a local-only system. Because of that, you need to make use of third-party software like NFS or GlusterFS. The big downfall with NFS is it's not encrypted. So for many businesses and developers, GlusterFS is the way to go.

I want to walk you through the process of using GlusterFS to share persistent storage in a Docker Swarm.

What You'll Need

I'll be demonstrating on a small cluster with one master and two nodes, each of which will be running on Ubuntu Server 18.04. So for that, you'll need:

- Three running and updated instances of Ubuntu Server 18.04.
- A user with **sudo** privileges.

That's all you need to make this work.

Update/Upgrade

Before you get going, it's always best to update and upgrade your server OS. To do this on Ubuntu (or any Debian-based platform), open a terminal and issue the commands:

TRENDING STORIES

1. Docker Basics: How to Use Dockerfiles
2. Exploring MicroOS, OpenSUSE's Immutable Container OS
3. What Is the Docker .env File and How Do You Use It?
4. Canonical Offers LTS 'Distroless' Containerized Apps for K8s
5. Container Image Fault Lines Are Being Exposed

```
sudo apt-get update
```

```
sudo apt-get upgrade -y
```

Should your kernel upgrade in the process, make sure to reboot the server so the changes will take effect.

Add Your Hosts

We now need to map our IP addresses in **/etc/hosts**. Do this on each machine. Issue the command:

```
sudo nano /etc/hosts
```

In that file (on each machine), you'll add something like this to the bottom of the file:

1 2 3	192.168.1.67 docker-master 192.168.1.107 docker-node1 192.168.1.117 docker-node2

Make sure to edit the above to match your IP addresses and hostnames.

Save and close the file.

Deploy the Swarm

If you haven't already done so, you need to install and deploy the Docker Swarm. On each machine install Docker with the command:

```
sudo apt-get install docker.io -y
```

Start and enable Docker with the commands:

```
sudo systemctl start docker
```

```
sudo systemctl enable docker
```

Add your user to the docker group (on all machines) with the command:

```
sudo usermod -aG docker $USER
```

Issue the following command (on all machines) so the changes take effect:

```
sudo newgrp docker
```

Next, we need to initialize the swarm. On the master issue the command:

```
docker swarm init --advertise-addr MASTER_IP
```

Where MASTER_IP is the IP address of the master.

Once the swarm has been initialized, it'll display the command you need to run on each node. That command will look like:

```
docker swarm join --token SWMTKN-1-09c0p3304ookcnibhg3lp5ovkjnylmxwjac9j5puvsj2wjzhn1-2vw4t2474ww1mbq4xzqpg0cru 192.168.1.67:2377
```

Copy that command and paste it into the terminal window of the nodes to join them to the master.

And that's all there is to deploying the swarm.

Installing GlusterFS

You now need to install GlusterFS on each server within the swarm. First, install the necessary dependencies with the command:

```
sudo apt-get install software-properties-common -y
```

Next, add the necessary repository with the command:

```
sudo add-apt-repository ppa:gluster/glusterfs-3.12
```

Update apt with the command:

```
sudo apt-get update
```

Install the GlusterFS server with the command:

```
sudo apt install glusterfs-server -y
```

Finally, start and enable GlusterFS with the commands:

```
sudo systemctl start glusterd
```

```
sudo systemctl enable glusterd
```

Generate SSH Keys

If you haven’t already done so, you should generate an SSH key for each machine. To do this, issue the command:

```
ssh-keygen -t rsa
```

Once you’ve taken care of that, it’s time to continue on.

Probing the Nodes

Now we’re going to have Gluster probe all of the nodes. This will be done from the master. I’m going to stick with my example of two nodes, which are docker-node1 and docker-node2. Before you issue the command, you’ll need to change to the superuser with:

```
sudo -s
```

If you don’t issue the Gluster probe command from root, you’ll get an error that it cannot write to the logs. The probe command looks like:

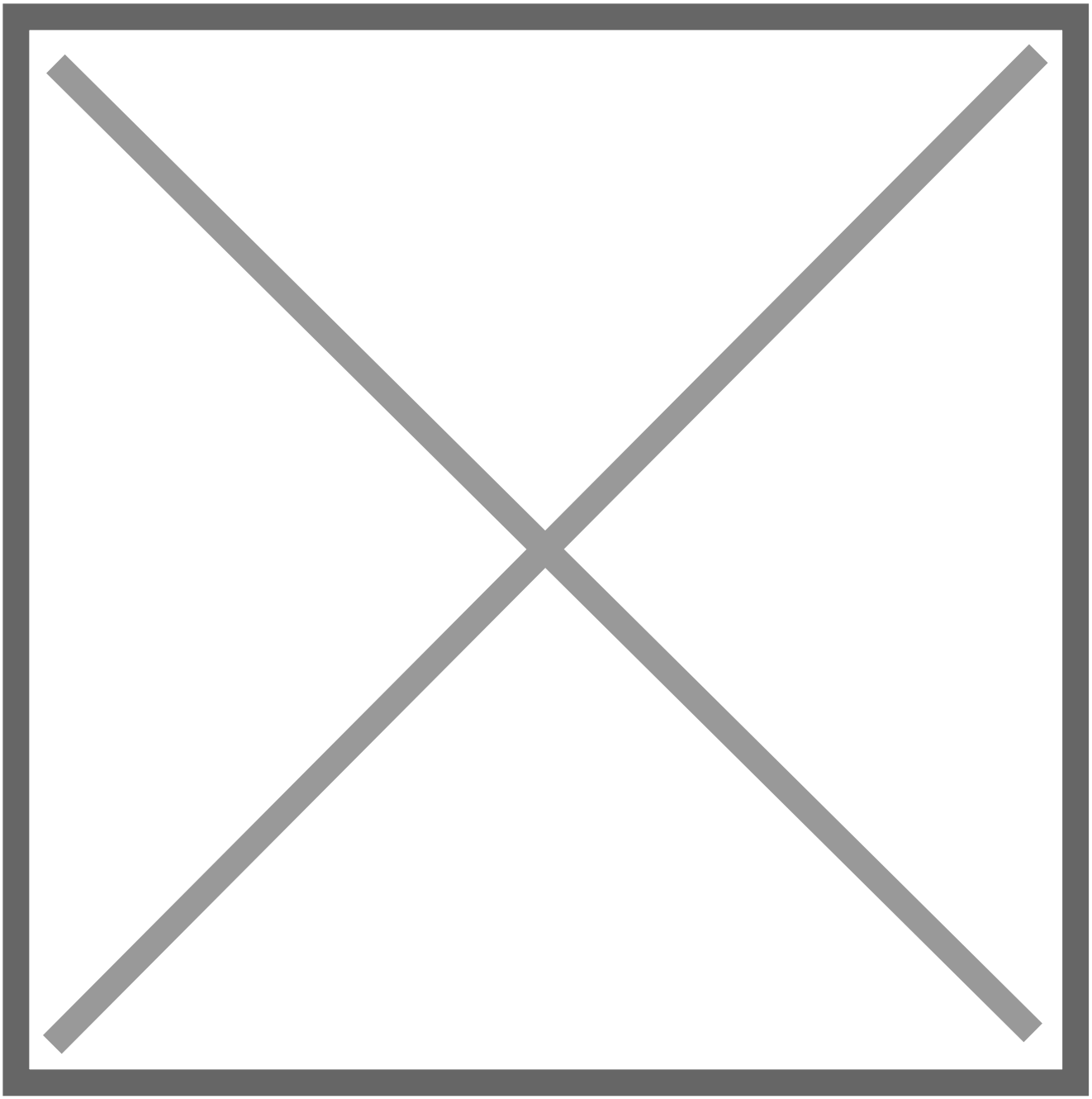
```
gluster peer probe docker-node1; gluster peer probe docker-node2;
```

Make sure to edit the command to fit your configuration (for hostnames).

Once the command completes, you can check to make sure your nodes are connected with the command:

```
gluster pool list
```

You should see all nodes listed as connected (**Figure 1**).



Zoom

Figure 1: Our nodes are connected.

Exit out of the root user with the **exit** command.

Create the Gluster Volume

Let’s create a directory to be used for the Gluster volume. This same command will be run on all machines:

```
sudo mkdir -p /gluster/volume1
```

Use whatever name you want in place of volume1.

Now we’ll create the volume across the cluster with the command (run only on the master):

```
sudo gluster volume create staging-gfs replica 3 docker-master:/gluster/volume1 docker-node1:/gluster/volume1 docker-node2:/gluster/volume1 force
```

Start the volume with the command:

```
sudo gluster volume start staging-gfs
```

The volume is now up and running, but we need to make sure the volume will mount on a reboot (or other circumstances). We’ll mount the volume to the /mnt directory. To do this, issue the following commands on all machines:

```
sudo -s
```

```
echo 'localhost:/staging-gfs /mnt glusterfs defaults,_netdev,backupvolfile-server=localhost 0 0' >> /etc/fstab
```

```
mount.glusterfs localhost:/staging-gfs /mnt
```

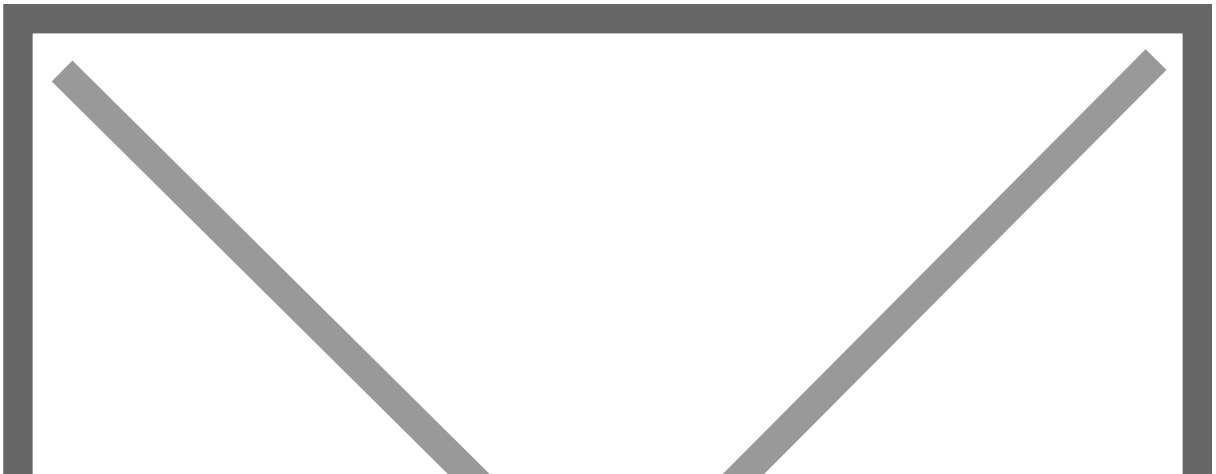
```
chown -R root:docker /mnt
```

```
exit
```

To make sure the Gluster volume is mounted, issue the command:

```
df -h
```

You should see it listed at the bottom (**Figure 2**).



Zoom

Figure 2: Our Gluster volume is mounted properly.

You can now create new files in the **/mnt** directory and they'll show up in the **/gluster/volume1** directories on every machine.

Using Your New Gluster Volume with Docker

At this point, you are ready to integrate your persistent storage volume with docker. Say, for instance, you need persistent storage for a MySQL database. In your docker YAML files, you could add a section like so:

```
1
2
3
4
```

```
<i> volumes:
</i><i>   - type: bind
</i><i>     source: /mnt/staging_mysql
</i><i>     target: /opt/mysql/data</i>
```

Since we've mounted our persistent storage in **/mnt** everything saved there on one docker node will sync with all other nodes.

And that's how you can create persistent storage and then use it within a Docker Swarm cluster. Of course, this isn't the only way to make persistent storage work, but it is one of the easiest (and cheapest). Give GlusterFS a try as your persistent storage option and see if it doesn't work out for you.

More Storage Tutorials

Tutorial: Set up Cloud Storage on a Linux Server, Using Seafile

Tutorial: Deploy a Highly Availability GlusterFS Storage Cluster

Tutorial: Dynamic Provisioning of Persistent Storage in Kubernetes with MiniKube

Tutorial: Deploy a Highly Availability GlusterFS Storage Cluster

Link: https://thenewstack.io/tutorial-deploy-a-highly-availability-glusterfs-storage-cluster/

The GlusterFS network file system is perfectly suited for various use cases that require the handling of large amounts (think petabytes) of stored data.

Nov 6th, 2020 10:32am by Jack Wallen
Featued image for: Tutorial: Deploy a Highly Availability GlusterFS Storage Cluster

The GlusterFS network file system is perfectly suited for various use cases that require the handling of large amounts (think petabytes) of stored data. In other words, this could be the ideal storage system for your various cloud or container deployments. With features like sharding, tiering, AFR Statistics, file snapshots, distributed hash tables, nonuniform file access, OVirt and QEMU integration, RDMA connection manager, rebalance, server quorum, distributed geo-replication, and brick failure detection, this file system might be ideal for your needs. Red Hat currently manages this open source network file system.

Of course, how you use GlusterFS with your cloud implementation will depend on which cloud platform you are using. But before you can roll it into your system, you first must get this networkable storage up and running.

I'm going to walk you through the process of deploying a three-node GlusterFS cluster on Ubuntu Server 20.04. To make this work you'll need three instances of Ubuntu. For my purposes those will have the following hostnames and IP addresses:

```
192.168.1.24 gluster1
```

```
192.168.1.25 gluster2
```

```
192.168.1.26 gluster3
```

You will want to change the IP addresses to match your network topography.

TRENDING STORIES

1. The Architect’s Guide to Storage for AI
2. The Architect’s Guide: A Modern Data Lake Reference Architecture
3. How to Work with Containers in TrueNAS
4. How to Create an Object Storage Bucket with MinIO Object Storage
5. Enable End-to-End Encryption Between Nextcloud and Your Desktop Client

First Steps

After spinning up your instances of Ubuntu, the first thing you want to do is update and upgrade each. You can do that (on all three) with the following two commands:

```
sudo apt-get update
```

```
sudo apt-get upgrade -y
```

If the kernel gets upgraded in any of these instances, you’ll want to make sure to reboot the server (so the updates get applied).

After you’ve upgraded, you’ll then want to set the hostname for each. This can be done with a handy command like so:

```
sudo hostnamectl set-hostname NAME
```

Where NAME will be gluster1, gluster2, and gluster3.

Next, we need to map the addresses in the /etc/hosts file. Open that file (on each server) for editing with the command:

```
sudo nano /etc/hosts
```

Map those addresses by adding the following at the bottom of the file:

```
1
2
3
```

```
192.168.1.24 gluster1
192.168.1.25 gluster2
192.168.1.26 gluster3
```

Save and close the file.

Installing GlusterFS

With the release of Ubuntu Server 20.04, GlusterFS is now found in the standard repositories. So to install the software, go back to the terminal window and issue the command:

```
sudo apt-get install glusterfs-server -y
```

Make sure to install GlusterFS on gluster1 and gluster2.

After the installation completes, start and enable GlusterFS on each server with the following two commands:

```
sudo systemctl start glusterd
```

```
sudo systemctl enable glusterd
```

Configuring GlusterFS

Now that your servers are ready and GlusterFS is installed, it’s time to configure gluster. On gluster1, create a trusted pool with the command:

```
sudo gluster peer probe gluster2
```

You should see peer probe: success returned.

Verify the status of the two peers with the command:

```
sudo gluster peer status
```

You should see that gluster2 is connected (Figure 1).

Figure 1: Our gluster1 and gluster2 servers are connected.

Zoom

Figure 1: Our gluster1 and gluster2 servers are connected.

Creating a Distributed Volume

We'll next create a distributed volume. I would highly recommend you create this volume on a partition that isn't within the system directory (aka, not on the same drive that your OS is hosted on). If you create this volume on the same drive as the OS, you could run into sync errors.

Let's create a new directory for GlusterFS (on both gluster1 and gluster2) with the command:

```
sudo mkdir -p /glusterfs/distributed
```

With the directory created, we can now create the volume (named v01) that will replicate on both gluster1 and gluster2. The command for this is:

```
sudo gluster volume create v01 replica 2 transport tcp gluster1:/glusterfs/distributed gluster2:/glusterfs/distributed
```

You will be prompted to okay the creation. Type "y" to allow the creation of the new distributed volume. Once that succeeds, start the volume with the command:

```
sudo gluster volume start v01
```

You can verify the creation with the command:

```
sudo gluster volume info v01
```

Installing the GlusterFS Client and Connecting to the Distributed Volume

It's now time to install the GlusterFS client. We'll do this on gluster3. For this, issue the command:

```
sudo apt install glusterfs-client -y
```

Create a new mount point for GlusterFS on gluster3 with the command:

```
sudo mkdir -p /mnt/glusterfs
```

We can now mount the distributed file system with the command:

```
sudo mount -t glusterfs gluster1:/v01 /mnt/glusterfs/
```

Finally, you'll want to make sure the distributed file system is mounted at boot. To do this, you'll need to edit the fstab file with the command:

```
sudo nano /etc/fstab
```

At the bottom of that file, add the following:

```
gluster1:/v01 /mnt/glusterfs glusterfs defaults,_netdev 0 0
```

Testing the Filesystem

With all of this in place, we can now test the GlusterFS distributed file system. On gluster1 issue the command:

```
sudo mount -t glusterfs gluster1:/v01 /mnt
```

On gluster2 issue the command:

```
sudo mount -t glusterfs gluster2:/v01 /mnt
```

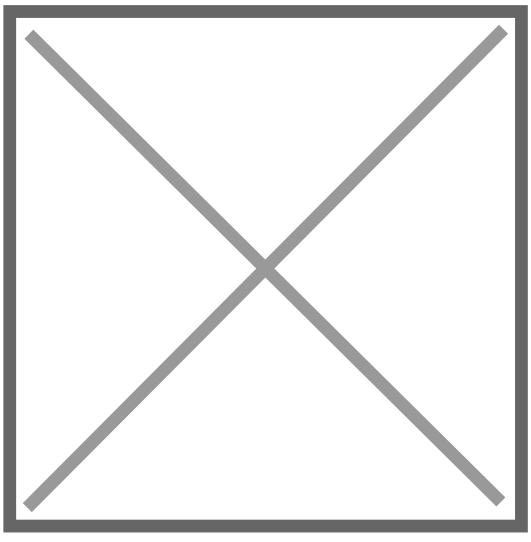
Move over to gluster3 and create a test file with the command:

```
sudo touch /mnt/glusterfs/thenewstack
```

Check to make sure the new file appears on both gluster1 and gluster2 with the command (run on gluster1 and gluster2):

```
ls /mnt
```

You should see thenewstack appear in both directories on gluster1 and gluster2 (Figure 2).



Zoom

Figure 2: The test file has appeared on gluster1.

And there you go, you now have a GlusterFS distributed file system up and running. You should now be able to integrate this into anything that requires a high-volume file system that offers plenty of features that can satisfy many of your cloud and container needs.

More Storage Tutorials

Tutorial: Set up Cloud Storage on a Linux Server, Using Seafile

Tutorial: Create a Docker Swarm with Persistent Storage Using GlusterFS

Tutorial: Dynamic Provisioning of Persistent Storage in Kubernetes with MiniKube

Setup a scalable high availability GlusterFS network filesystem with Docker Swarm on Ubuntu Server 20.04 LTS

Link: https://florianmuller.com/setup-a-scalable-high-availability-glusterfs-network-filesystem-with-docker-swarm-on-ubuntu-server-20-04-lts

In this quick guide we are going to setup the scalable GlusterFS filesystem for a four node Docker Swarm cluster on Ubuntu 20.04 LTS. More precise: we replace an existing persistent NFS storage on the cluster with the new GlusterFS and make it available under the same old mount point as the old NFS. Therefore we can easily redeploy all of our containers from our docker-compose files, without changing anything. Lets get started!

Install GlusterFS

First we ssh into our master node of our docker swarm, in our case `pi-cluster-1`. There we update our repos and install GlusterFS together with its necessary dependencies:

```
sudo apt update && sudo apt install software-properties-common glusterfs-server -y
```

Now finish and start the GlusterFS service:

```
sudo systemctl start glusterd
sudo systemctl enable glusterd
```

Amazing! **Now repeat these install steps on ALL 3 remaining nodes!**

Prepare ufw firewall (if set):

If you are using a firewall on your ubuntu nodes, which I highly recommend, you need to open some ports to give GlusterFS a chance to communicate with the nodes. In our case we are using the built in ufw, **but we are NOT adding the standard ports itself, as glusterfs is generating new ports per bricks on a glusterfs volume, that can give you lots of problems later on.**

Instead we are allowing on every node explicitly the other nodes to be incoming traffic without restrictions, so you do not need to worry about future port changes by glusterfs. **Add the following rules ONLY on the cluster they are meant for. not on any others (replace with your cluster IP addresses):**

```
#cluster1 (192.168.0.1)
sudo ufw allow proto any from 192.168.0.2 to 192.168.0.1
sudo ufw allow proto any from 192.168.0.3 to 192.168.0.1
sudo ufw allow proto any from 192.168.0.4 to 192.168.0.1

#cluster2 (192.168.0.2)
sudo ufw allow proto any from 192.168.0.1 to 192.168.0.2
sudo ufw allow proto any from 192.168.0.3 to 192.168.0.2
sudo ufw allow proto any from 192.168.0.4 to 192.168.0.2

#cluster3 (192.168.0.3)
sudo ufw allow proto any from 192.168.0.1 to 192.168.0.3
sudo ufw allow proto any from 192.168.0.2 to 192.168.0.3
sudo ufw allow proto any from 192.168.0.4 to 192.168.0.3

#cluster4 (192.168.0.4)
sudo ufw allow proto any from 192.168.0.1 to 192.168.0.4
sudo ufw allow proto any from 192.168.0.2 to 192.168.0.4
sudo ufw allow proto any from 192.168.0.3 to 192.168.0.4
```

Repeat these firewall steps on ALL 3 remaining nodes, and use the rules applying to the node!

If, and only IF, you run into any issues (which you shouldn't) with glusterfs access or syncing or you get a split brain, than check out the specific ports per cluster as well, and see if you might need them. These are the glusterfs standard ports from their documentation:

```
#glusterfs
111 (portmapper)
24007/tcp
24008/tcp
49152 – 49155
```

Probe and connect the nodes to GlusterFS:

Now we're going to have Gluster probe all of the nodes. **This will be done from the master node.** I'm going to stick with my example of four nodes, which are these hostnames:
`pi-cluster-1`
`pi-cluster-2`
`pi-cluster-3`
`pi-cluster-4`
Before you issue the command, you'll need to change to the superuser with:

```
sudo -s
```

For some reason in my case the hostnames did not work and I had to use the local IP address of my nodes. So if you get a connection error from GlusterFS while probing, use the local IP address instead.

If you use hostnames probe like this (replace with YOUR nodes hostnames):

```
gluster peer probe pi-cluster-1; gluster peer probe pi-cluster-2; gluster peer probe pi-cluster-3; gluster peer probe pi-cluster-4;
```

When you want to use hostnames (recommended) and you get an error at first, try to lookup and set your hostnames in /etc/hosts on each cluster:

```
sudo nano /etc/hosts
```

Edit the host file like this, but remember to replace with your cluster nodes IP addresses:

```
127.0.0.1 localhost
192.168.0.1 pi-cluster-1
192.168.0.2 pi-cluster-2
192.168.0.3 pi-cluster-3
```

```
192.168.0.4 pi-cluster-4
```

If you still have issues probing, you can also use direct local IP addresses (not recommended, but works) probe like this (replace with YOUR nodes IP addresses):

```
gluster peer probe 192.168.0.1; gluster peer probe 192.168.0.2; gluster peer probe 192.168.0.3; gluster peer probe 192.168.0.4;
```

Great! Now you have successfully added the nodes to each other in Gluster. To verify your Gluster pool, type this on the Master node:

```
gluster pool list
```

If you get a list of your nodes you are good to go and should exit the superuser mode with:

```
exit
```

Create the GlusterFS Volume on your drives:

Now we create our GlusterFS volume mount point. I called mine dockerfiles, you can choose whatever you want here:

```
sudo mkdir -p /gluster/dockerfiles
```

You need to run and create this mount point on ALL nodes and machines!

Once done, you can create the volume across the cluster with this command (**run only once on the master**):

If you used hostnames before:

```
sudo gluster volume create staging-gfs replica 4 pi-cluster-1:/gluster/dockerfiles pi-cluster-2:/gluster/dockerfiles pi-cluster-3:/gluster/dockerfiles pi-cluster-4:/gluster/dockerfiles force
```

If you used IP addresses before:

```
sudo gluster volume create staging-gfs replica 4 192.168.0.1:/gluster/dockerfiles
192.168.0.2:/gluster/dockerfiles 192.168.0.3:/gluster/dockerfiles
192.168.0.4:/gluster/dockerfiles force
```

Startup the Gluster volume with (on the master node only):

```
sudo gluster volume start staging-gfs
```

The volume is now up and running, but we need to make sure the volume will mount on the same mount point as our old NFS storage. We also need to migrate our container data from the prior NFS storage to the new glusterFS volume first.

Migrate container data from NFS:

Create the directory where we will have our shared GlusterFS volume mounted to.
IMPORTANT if you migrate from NFS: This should be a temporary name first, as we copy back and forth from it. So for our case where we migrate we create a temporary mount point with:

```
#if not exists:
sudo mkdir -p /mnt/docker
sudo mkdir -p /mnt/tempdockerfiles
```

Next we mount our GlusterFS volume to it:

```
sudo mount.glusterfs localhost:/staging-gfs /mnt/tempdockerfiles
```

Perfect! Now we copy and migrate our docker container data over to GlusterFS.
IMPORTANT: If you have write sensitive data/services, you need to stop and remove first all our stacks and services. Get a list of what running on your stack with: `docker stack ls`

Now lets copy our existing NFS docker data from our actual docker used shared volume /mnt/docker to our temp GlusterFS mount point:

```
sudo cp -a /mnt/docker/. /mnt/tempdockerfiles/
```

This may take a while, but once finished you should have all your actuall container data under /mnt/tempdockerfiles as well. You can verify this with: `ls -l /mnt/tempdockerfiles`

Once you are sure all data has been copied, we unmount our NFS storage volume and our GlusterFS with. **Run the following commands on ALL FOUR NODES**:

```
sudo umount /mnt/docker/
sudo umount /mnt/tempdockerfiles/
```

Make sure they are unmounted with `ls -l /mnt/tempdockerfiles/` that should give you an empty result (0 files).

As we have copied our docker data to the GlusterFS volume we can also delete the temporary mount point now (**run on all nodes**):

```
rm -R /mnt/tempdockerfiles/
```

Now we mount the GlusterFS volume in the old mount point of our prior NFS storage (**run on all nodes**):

```
sudo mount.glusterfs localhost:/staging-gfs /mnt/docker
```

Verify with `ls -l /mnt/docker` and you should see all your old docker container data again. last but not least we need to set the permissions right on the volume and make the mount persistent over a reboot (or other circumstances):

Set permissions (**on all nodes**):

```
### replace ubuntu with your username
sudo chown -R ubuntu:ubuntu /mnt/docker/
sudo chown root:docker /mnt/docker/
```

Add mount point to /etc/fstab (**on all nodes**):

```
echo 'localhost:/staging-gfs /mnt/docker glusterfs defaults,_netdev,backupvolfile-
server=localhost 0 0' >> /etc/fstab
```

Important: If you had an old entry in /etc/fstab from auto mounting your NFS shared volume on boot to /mnt/docker, you need to remove that line from fstab now as well. (on all nodes)

Now reboot ALL NODES with:

```
sudo reboot now
```

After the reboot you can verify if the GlusterFS volume is mounted correctly on your nodes with:

```
df -h
```

Great! Thats it, you successfully installed GlusterFS on your Docker Swarm cluster and migrated all your container data from the NFS shared volume over the the Gluster volume!

Appendix: Complete uninstall glusterfs

If you at any point need to fully uninstall and remove glusterfs from all your nodes, you should to the following on all nodes:

```
#enter root
sudo -s

gluster volume stop staging-gfs
gluster volume delete staging-gfs

gluster peer detach 192.168.0.4
gluster peer detach 192.168.0.3
gluster peer detach 192.168.0.2
gluster peer detach 192.168.0.1

apt --purge remove glusterfs-server -y
apt autoremove -y

rm -rf /gluster/
rm -rf /var/log/glusterfs/
rm -rf /var/lib/glusterd/
rm -rf /run/gluster/
rm -rf /usr/lib/python3/dist-packages/gluster/



#Remove nano /etc/fstab boot mount entry for gluster:
nano /etc/fstab

#Remove all firewall rules as above:
ufw delete <position count of your rule in ufw status>
```

After that you should restart each node and you are back to a state where you could reinstall glusterfs fresh.

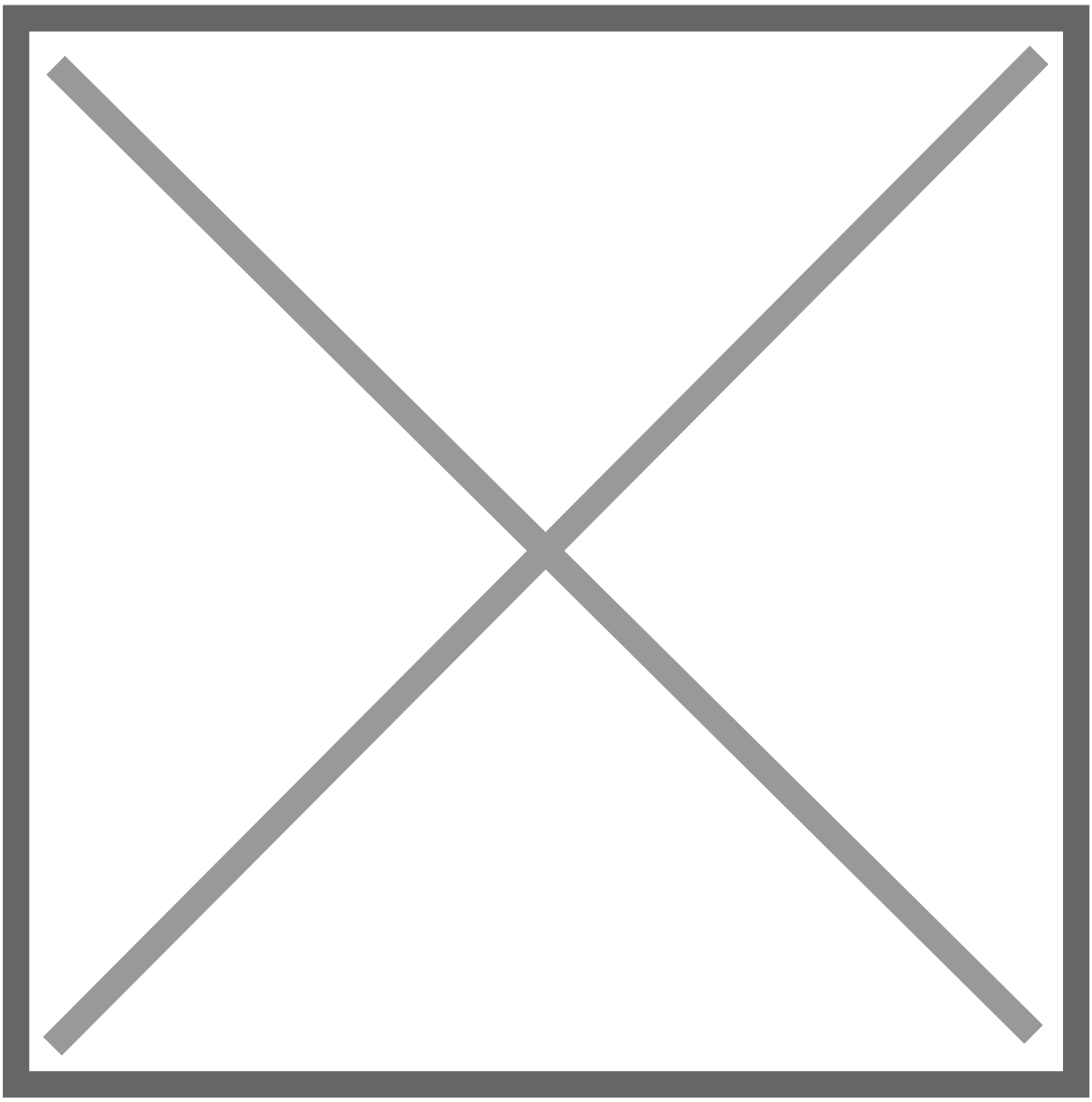
Published by Florian Müller in Development, Technology

Tags: docker, docker swarm, docker-compose, filesystem, gluster, glusterfs, ssh, terminal, ubuntu

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.

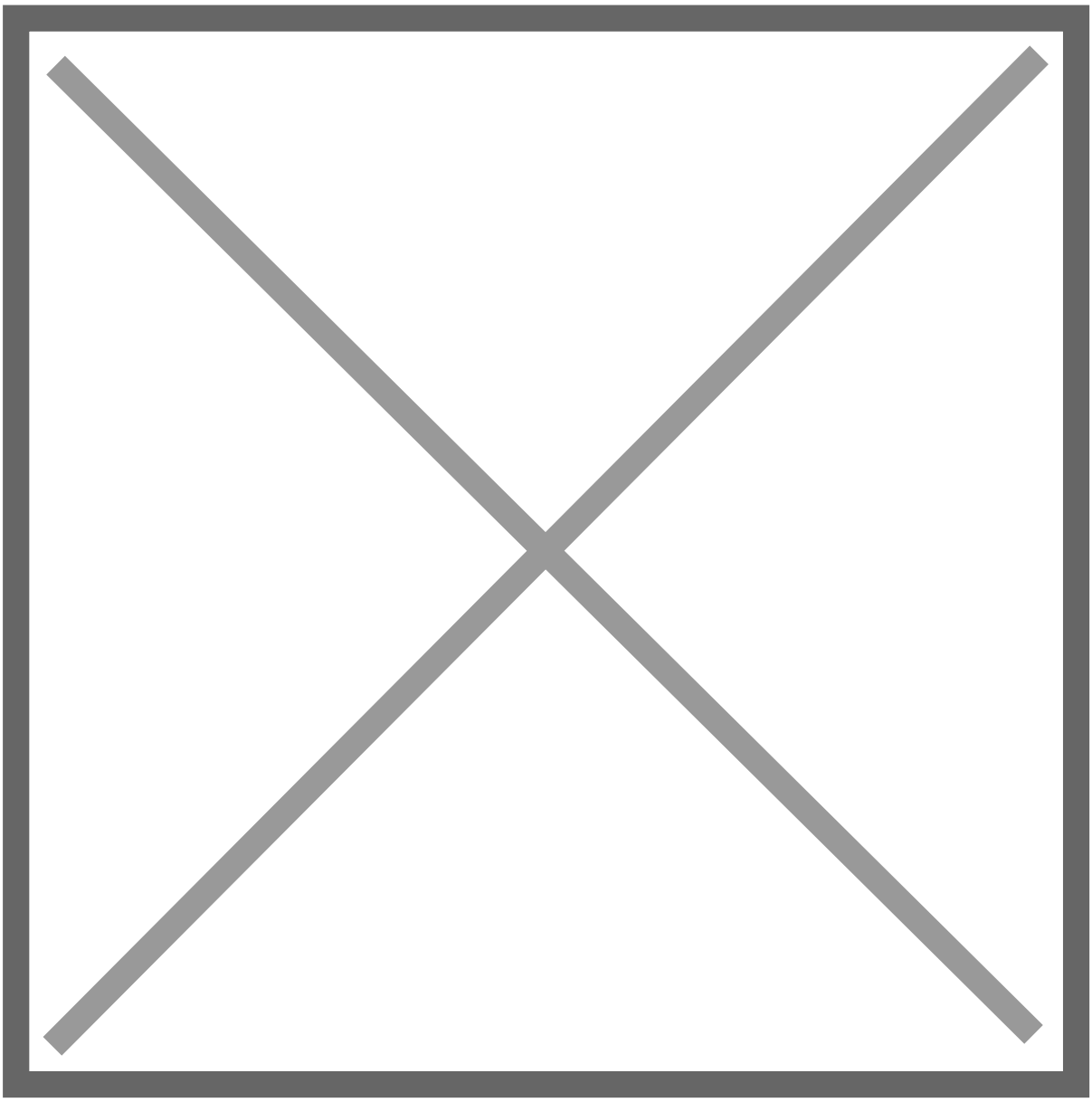
```
swarm-manager:~# cat wordpress-stack.yml
# wordpress-stack.yml
version: '3.1'services:  wordpress:
    image: wordpress
    restart: always
    ports:
      - 8080:80
    environment:
      WORDPRESS_DB_HOST: db
      WORDPRESS_DB_USER: exampleuser
```

```
      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.

Configuração GlusterFS Docker

ChatGPT

sim

O ChatGPT disse:

Ótimo! Abaixo está o **passo a passo completo** para instalar o **GlusterFS no host** e integrá-lo ao **Jibri (do Jitsi Meet) via Docker**, com foco em **compartilhar e replicar o diretório de gravações** entre servidores.

---

1. Instalar o GlusterFS em cada host

Execute nos 3 servidores: `server00`, `server01`, `server04`

Para Ubuntu 20.04/22.04:

```
sudo apt update
sudo apt install -y glusterfs-server
sudo systemctl enable --now glusterd
```

Para CentOS/RHEL:

```
sudo yum install -y centos-release-gluster
sudo yum install -y glusterfs-server
sudo systemctl enable --now glusterd
```

🔧 2. Configurar resolução de nomes ou editar `/etc/hosts`

Edite o arquivo `/etc/hosts` em todos os 3 servidores:

```
147.93.176.49 server00
147.93.176.51 server01

147.93.176.48 server04
```

🔗 3. Formar o cluster GlusterFS (em `server00`)

```
sudo gluster peer probe server01

sudo gluster peer probe server04
```

Verifique:
`sudo gluster peer status`

📁 4. Criar os diretórios dos bricks

Execute:
`sudo mkdir -p /glusterfs/gravacoes_jitsi`

---

5. Criar volume replicado (em `server00`)

```
sudo gluster volume create jitsi-recordings \
  replica 3 \
  server00:/glusterfs/gravacoes_jitsi \
  server01:/glusterfs/gravacoes_jitsi \
  server04:/glusterfs/gravacoes_jitsi
```

Iniciar volume:
`sudo gluster volume start jitsi-recordings`

Verificar:
`sudo gluster volume info`

---

6. Montar o volume GlusterFS nos hosts

Crie ponto de montagem:
`sudo mkdir -p /opt/aplicativos/servico01/volumes`

Monte o volume do gfs no diretório que deseja fazer replicação:
`sudo mount -t glusterfs server00:/jitsi-recordings /opt/aplicativos/servico01/volumes`

Para montagem automática via `/etc/fstab`:
`echo "server00:/jitsi-recordings /opt/aplicativos/servico01/volumes glusterfs defaults,_netdev 0 0" | sudo tee -a /etc/fstab`

🔗 7. Integrar com Docker Jibri

Altere o `docker-compose.override.yml` ou diretamente no `docker-compose.yml` do Jibri:

```
services:

  jibri:

    volumes:
      - ./volumes/jibri:/config/recordings
```

🧪 8. Testar

1. Reinicie o Jibri: `docker-compose up -d jibri`
2. Inicie uma gravação no Jitsi
3. Verifique se o vídeo aparece em `/glusterfs/gravacoes_jitsi`
   Veja se aparece em todos os nós (replicação automática)

🛠️ Diagnóstico

Comandos úteis:

```
gluster volume status
```

```
gluster volume heal jitsi-recordings info
```

```
gluster peer status
```

Considerações finais

- Evite usar GlusterFS dentro de containers para dados críticos.
- Mantenha os servidores sincronizados com NTP.
- Evite desligar servidores abruptamente (pode causar split-brain).

Configuração GlusterFS Docker2

ChatGPT

Perfeito! Vamos criar uma configuração de **GlusterFS replicada entre os três servidores**:

- `server04` → `147.93.176.48`
- `server02` → `212.56.34.241`
- `server03` → `212.56.34.172`

Com os diretórios locais:

swift
```
/opt/swarm/npm/volumes/data
/opt/swarm/npm/volumes/letsencrypt
/opt/swarm/npm/volumes/mysql
```

Você quer que **cada um desses diretórios seja um volume replicado entre os três nós** — ou seja, 3 volumes distintos replicados 3x. Vamos fazer isso em etapas bem claras.

---

1. Configure `/etc/hosts` (em **todos os servidores**)

Para facilitar o uso de nomes ao invés de IPs diretos:

bash
```
echo "147.93.176.48 server04" | sudo tee -a /etc/hosts
echo "212.56.34.241 server02" | sudo tee -a /etc/hosts
echo "212.56.34.172 server03" | sudo tee -a /etc/hosts
```

🔧 2. Instale o GlusterFS (em todos os servidores)

Ubuntu/Debian:

bash
```
sudo apt update
sudo apt install glusterfs-server -y
sudo systemctl enable --now glusterd
```

CentOS/RHEL:

bash
```
sudo yum install centos-release-gluster
sudo yum install glusterfs-server -y
sudo systemctl enable --now glusterd
```

🔧 3. Verifique a conectividade entre os nós

Em todos os servidores, teste:

bash
```
ping server02
ping server03
ping server04
```

🔧 4. No `server04`: pareie os outros nós

No **server04**:

bash
```
gluster peer probe server02
gluster peer probe server03
```
Verifique:

bash
```
gluster peer status
```
Todos devem estar com status: `Peer in Cluster (Connected)`

---

5. Crie os diretórios de dados (em todos os 3 nós)

bash
```
sudo mkdir -p /opt/swarm/npm/volumes/data
sudo mkdir -p /opt/swarm/npm/volumes/letsencrypt
sudo mkdir -p /opt/swarm/npm/volumes/mysql
```
Recomendo garantir permissões para UID 1000 (usado por containers):

bash
```
sudo chown -R 1000:1000 /opt/swarm/npm/volumes
```
---

6. No `server04`: crie os volumes GlusterFS replicados

Volume `npm-data`:

bash
```
gluster volume create npm-data replica 3 \
  server04:/opt/swarm/npm/volumes/data \
  server02:/opt/swarm/npm/volumes/data \
  server03:/opt/swarm/npm/volumes/data force

gluster volume start npm-data
```
Volume `npm-letsencrypt`:

bash

```
gluster volume create npm-letsencrypt replica 3 \
  server04:/opt/swarm/npm/volumes/letsencrypt \
  server02:/opt/swarm/npm/volumes/letsencrypt \
  server03:/opt/swarm/npm/volumes/letsencrypt force

gluster volume start npm-letsencrypt
```

□□ Volume `npm-mysql`:

bash

```
gluster volume create npm-mysql replica 3 \
  server04:/opt/swarm/npm/volumes/mysql \
  server02:/opt/swarm/npm/volumes/mysql \
  server03:/opt/swarm/npm/volumes/mysql force

gluster volume start npm-mysql
```

□ 7. Verifique se os volumes foram criados corretamente

bash

```
gluster volume info
```

Você deve ver os 3 volumes com `Status: Started` e `Number of Bricks: 3 x 1 = 3`.

□ 8. Monte os volumes no cliente Docker Swarm

No(s) nó(s) Swarm onde você quer usar os dados, monte os volumes:

bash

```
sudo mkdir -p /mnt/npm-swarm/data

sudo mkdir -p /mnt/npm-swarm/letsencrypt

sudo mkdir -p /mnt/npm-swarm/mysql
```

```
sudo mount -t glusterfs server04:/npm-data /mnt/npm-swarm/data

sudo mount -t glusterfs server04:/npm-letsencrypt /mnt/npm-swarm/letsencrypt

sudo mount -t glusterfs server04:/npm-mysql /mnt/npm-swarm/mysql
```

---

9. (Opcional) Automatize com `/etc/fstab`

Adicione ao `/etc/fstab` para montar automaticamente no boot:

fstab
```
server04:/npm-data         /mnt/npm-swarm/data         glusterfs defaults,_netdev 0 0

server04:/npm-letsencrypt  /mnt/npm-swarm/letsencrypt  glusterfs defaults,_netdev 0 0

server04:/npm-mysql        /mnt/npm-swarm/mysql        glusterfs defaults,_netdev 0 0
```

> Teste com: `sudo mount -a`

Conclusão

Você agora terá:

Volume GlusterFS	Diretório local montado	Usado por
`npm-data`	`/mnt/npm-swarm/data`	Nginx Proxy Mgr
`npm-letsencrypt`	`/mnt/npm-swarm/letsencrypt`	Nginx Proxy Mgr
`npm-mysql`	`/mnt/npm-swarm/mysql`	MariaDB

Se quiser, posso agora gerar seu `docker-stack.yml` já com esses volumes configurados para Swarm. Deseja?