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:


```
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:

```
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:

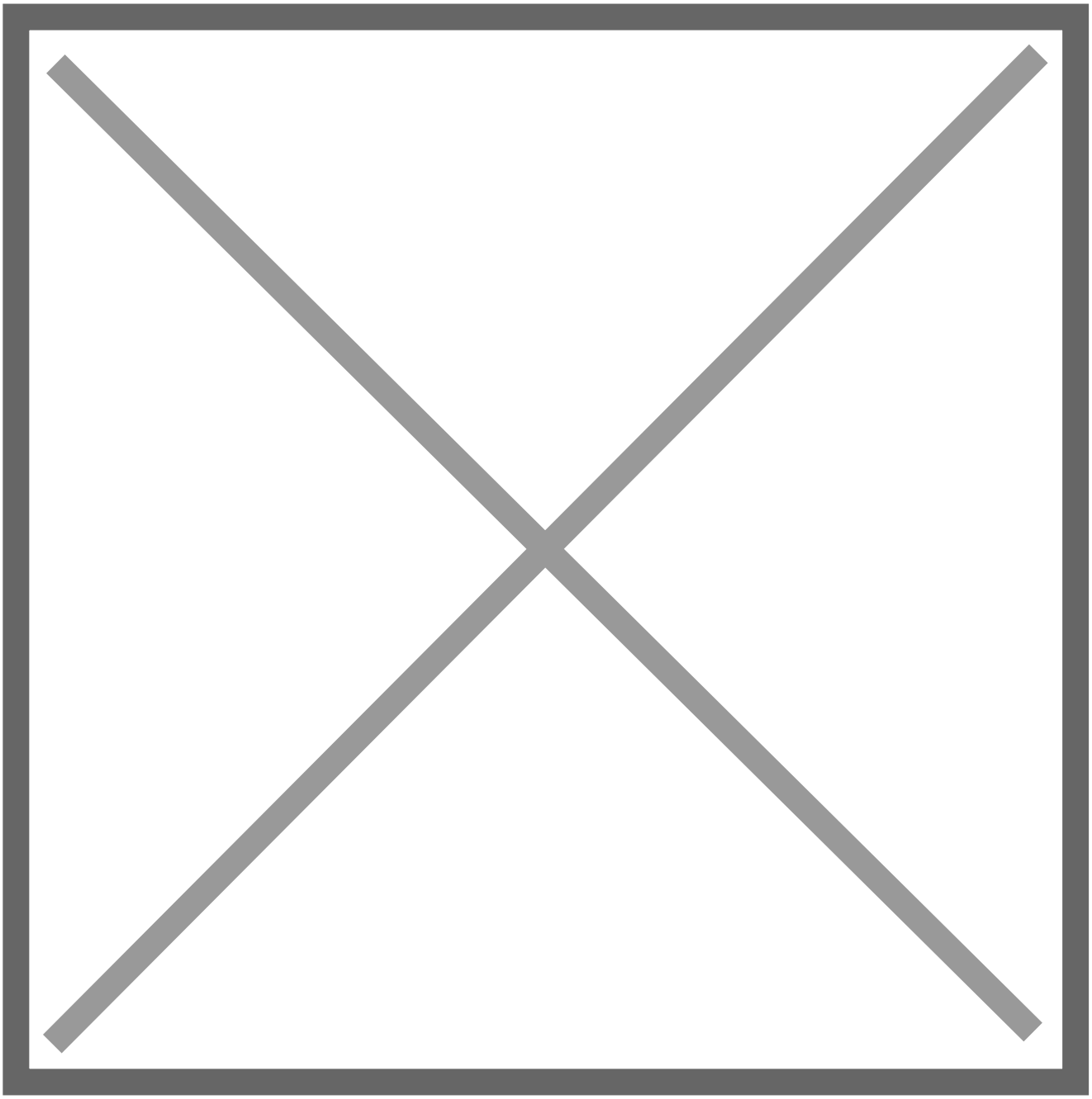
```
gluster peer probe docker-node1; gluster peer probe docker-node2;
```

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
```

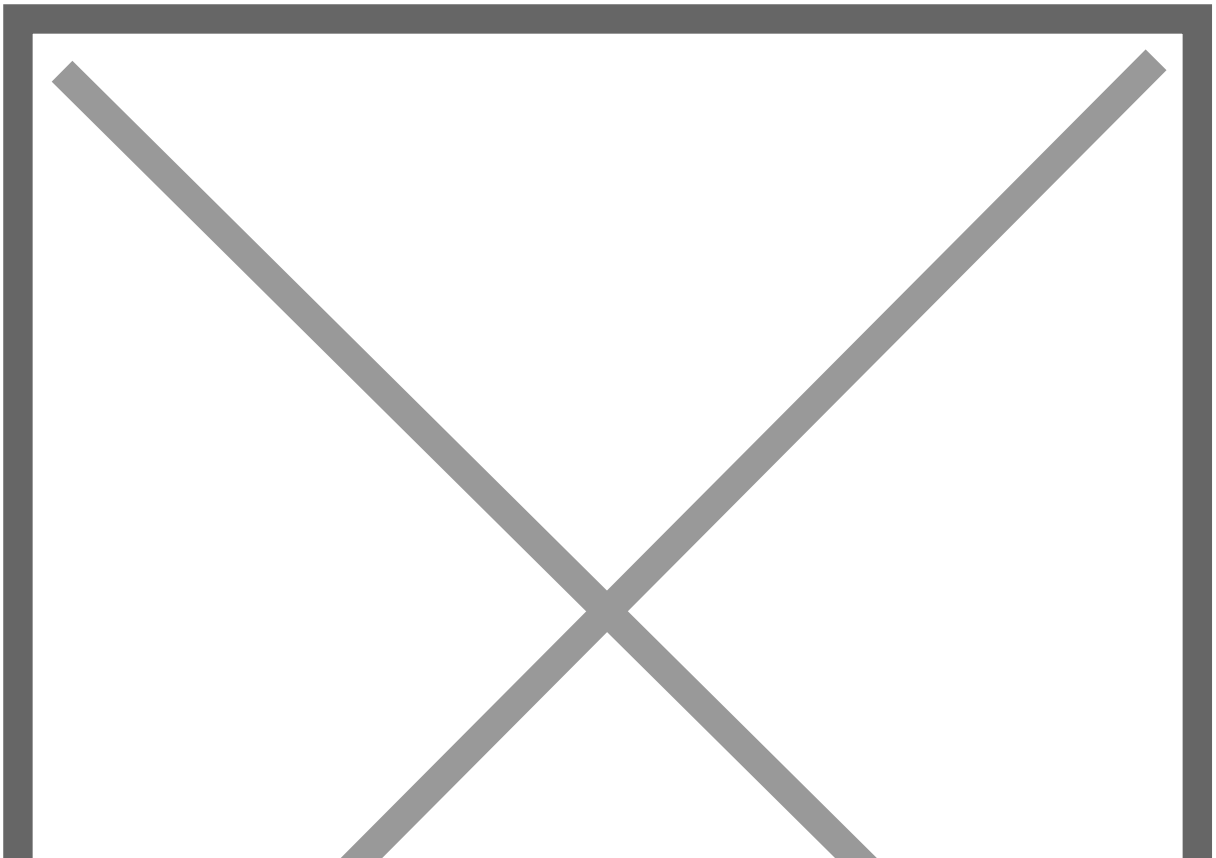
```
chown -R root:docker /mnt
```

```
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

---

Revision #1
Created 7 July 2024 17:26:29 by Administrador
Updated 7 July 2024 17:29:18 by Administrador