

GlusterFS & Keepalived Setup

Keep your Docker Swarm Files synced in Realtime.

- GlusterFS
 - What is GlusterFS?
 - Step-by-Step Guide: Setup GlusterFS
 - Step-by-Step Guide: How to Mount GlusterFS on Boot
- Keepalived
 - What is Keepalived?
 - Step-by-Step Guide: Setup Keepalived

GlusterFS

What is GlusterFS?

GlusterFS Basic Explanation

GlusterFS is an open-source distributed file system that allows you to pool storage resources from multiple servers (nodes) into a single file system. It's designed to handle large amounts of data by distributing it across many machines, making it scalable and fault-tolerant. Essentially, GlusterFS lets you combine the storage capacity of several machines into a shared system that all machines can access.

GlusterFS is managed by the `glusterd` service, which is the core of the system. It keeps track of which files are stored on which machines and ensures that data is replicated across different nodes. This replication is useful because it allows your data to survive even if one of your nodes fails, ensuring high availability.

Why Use GlusterFS for Syncing Docker Swarm Nodes?

When running services in Docker Swarm's replicated mode, the same service runs on multiple nodes to ensure high availability and scalability. These nodes may need to access the same set of data files, and that's where GlusterFS becomes very useful. Instead of storing separate copies of files on each node (which could lead to inconsistencies), GlusterFS ensures that all nodes share the same file system and stay in sync.

Using GlusterFS on the host system (rather than inside Docker) ensures that the file syncing works across the entire system, regardless of Docker's configuration. This setup is independent of the containers, allowing for seamless file sharing even if containers are recreated or moved to different nodes. Additionally, since GlusterFS operates at the system level, it can sync files across nodes more efficiently, avoiding network bottlenecks inside Docker containers.

Setting up GlusterFS across the nodes ensures that any file written on one node is immediately available to the others, providing data consistency and reliability when scaling services across a Docker Swarm cluster.

Step-by-Step Guide: Setup GlusterFS

If you dont know what GlusterFS is what what its for, you may consider check out this [Post](#).

Step-by-Step Guide to Setting Up GlusterFS on a 3-Node Cluster

This guide will walk you through setting up GlusterFS on a 3-node cluster using Raspberry Pis, with the IP addresses: `192.168.0.10`, `192.168.0.11`, and `192.168.0.12`. GlusterFS will be used to create a distributed, replicated file system across these nodes. Follow these steps to get your cluster up and running.

Step 1: Install GlusterFS on All Nodes

First, you need to install the required packages on **all nodes**. The `software-properties-common` package allows managing repositories, and `glusterfs-server` installs the GlusterFS server.

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

This ensures that every node in your cluster has GlusterFS installed and ready to share and replicate files.

Step 2: Enable Automatic Start of GlusterFS on Reboot

To ensure GlusterFS starts automatically after each reboot, you need to enable the `glusterd` service (GlusterFS daemon) on all nodes. Run this command on each node:

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

This command starts the Gluster service immediately and ensures it starts automatically on system reboot.

Step 3: Peer Probe the Nodes

Now, log in to the main node (192.168.0.10) via SSH as root. You need to "peer probe" the other nodes to add them to the GlusterFS cluster. The peer probe is a command that connects the other nodes to the Gluster network, allowing them to participate in the distributed file system.

On the main node (192.168.0.10), run the following commands:

```
gluster peer probe 192.168.0.11  
gluster peer probe 192.168.0.12
```

The `peer probe` command tells the main node to reach out to the other nodes, establishing a connection and syncing them into the cluster.

Step 4: Create the GlusterFS Volume

A volume in GlusterFS is essentially a storage pool made up of directories on different nodes. Here, we'll create a **replicated** volume across all three nodes. Replication ensures that the same data is stored on all nodes, making it resilient to failures.

Run the following command on the main node (192.168.0.10):

```
gluster volume create [glustername] replica 3 192.168.0.10:/root/gluster 192.168.0.11:/root/gluster  
192.168.0.12:/root/gluster force
```

Explanation:

- `[glustername]` is the name you want to give to your GlusterFS volume.
- `replica 3` specifies that this is a replicated volume across all 3 nodes.
- The paths after each node's IP specify where the volume will be stored on each node (`/root/gluster`).

The `force` option is used to bypass potential warnings (such as creating volumes in a root directory).

Step 5: Start the GlusterFS Volume

Once the volume is created, you need to start it. This also needs to be done on the main node:

```
gluster volume start [glustername]
```

Replace `[glustername]` with the name of the volume you created in the previous step.

Step 6: Create a Mount Directory on Each Node

On **each node**, create a directory where you will mount the GlusterFS volume. For this guide, we'll use `/mnt/glustermount` as the mount point.

Run this command on all nodes:

```
sudo mkdir -p /mnt/glustermount
```

This ensures that the GlusterFS volume has a place to be mounted on each node.

Step 6.1: Mount the GlusterFS Volume

Now, mount the GlusterFS volume on the `/mnt/glustermount` folder. Run this command on each node:

```
sudo mount.glusterfs localhost:/[glustername] /mnt/glustermount
```

Replace `[glustername]` with the name of your volume. This mounts the volume, making it accessible from the `/mnt/glustermount` directory on each node.

To Setup Automatic Mount of GlusterFS on Boot check out this [Post](#),

Your 3-node GlusterFS cluster should now be up and running, providing a replicated and distributed file system that's resilient and ready for use!

Step-by-Step Guide: How to Mount GlusterFS on Boot

To ensure that your GlusterFS volume is automatically mounted at boot, you'll need to make some adjustments. By default, GlusterFS volumes might not mount properly on startup due to timing issues, where the Gluster service (`glusterd`) hasn't fully started yet. The following steps provide a workaround that ensures your GlusterFS volume is mounted after boot.

This guide is based on insights from [this ServerFault discussion](#), modified to suit your 3-node Raspberry Pi setup.

Step 1: Modify the GlusterFS Service to Wait Before Mounting

First, you need to add a script that ensures the GlusterFS service has fully started before the system attempts to mount the volume.

Run the following command to edit the `glusterd.service`:

```
sudo systemctl edit glusterd.service
```

This will open a text editor. Add the following configuration to ensure a delay in mounting the GlusterFS volume:

```
[Service]
ExecStartPost=/usr/local/sbin/glusterfs-wait
```

Save and exit the editor.

Step 2: Create the `glusterfs-wait` Script

Next, create a script that checks if the GlusterFS volume is ready to be mounted. This script will be called after the Gluster service starts and will retry the mount command until it succeeds.

Create the script file:

```
sudo nano /usr/local/sbin/glusterfs-wait
```

Paste the following content into the file:

```
#!/bin/bash
FAIL=1
until [ $FAIL -eq 0 ]; do
    gluster volume status all
    FAIL=$?
    test $FAIL -ne 0 && sleep 1
done
exit 0
```

Save and close the file.

This script will keep checking the status of the GlusterFS volume every second until the volume is ready.

Step 3: Make the Script Executable

You need to make the script executable so it can run at startup.

Run this command:

```
sudo chmod +x /usr/local/sbin/glusterfs-wait
```

Step 4: Create a Mount Override

To ensure that the GlusterFS mount happens after the `glusterd.service` is up and running, you need to create a mount override.

For the example folder `/mnt/glustermount`, run the following command:

```
sudo systemctl edit mnt-glustermount.mount
```


In the text editor, paste the following content:

```
[Unit]
After=glusterd.service
Wants=glusterd.service
```

This ensures that the mount service will wait for the `glusterd` service to start.
Save and close the editor.

Step 5: Reload the Systemd Daemon and Reboot

To apply the changes, reload the systemd daemon with the following command:

```
sudo systemctl daemon-reload
```

Then, reboot your system:

```
sudo reboot
```

Step 6: Verify GlusterFS Mount After Reboot

After the system reboots, check the status of the GlusterFS service and ensure the volume is mounted correctly:

```
systemctl status glusterd.service
```

You should see that the GlusterFS volume has mounted successfully, and the service is running without issues.

This setup ensures that your GlusterFS volumes are reliably mounted after each system boot.

Keepalived

What is Keepalived?

Keepalived Basic Explanation

Keepalived is an open-source service commonly used to ensure high availability for network services. It helps by monitoring the health of services and automatically switching over to a backup server if the primary server goes down. This process is known as failover. Keepalived works by using a protocol called VRRP (Virtual Router Redundancy Protocol) to create a virtual IP address that is shared between multiple servers. The virtual IP always points to the currently active (healthy) server, ensuring that users can access services without interruption, even if one server fails.

Keepalived constantly monitors the health of the primary server. If a failure is detected, it assigns the virtual IP address to a backup server, which takes over, keeping the service running smoothly.

Why Use Keepalived for Syncing Docker Swarm Nodes?

In a Docker Swarm environment, you may want to ensure that your services are highly available even if a node goes down. This is where Keepalived becomes useful. By using Keepalived, you can provide a single, consistent virtual IP address that always points to the active node in your cluster. This means that users or external services only need to connect to one IP address, and they will always be directed to a healthy node.

When used on the host system rather than inside Docker, Keepalived can manage failovers at the network level. This ensures that the Docker Swarm services running in replicated mode remain accessible through a single virtual IP, even if the node hosting the service fails. It provides seamless failover without relying on Docker's internal mechanisms, giving more flexibility and reliability in how the swarm nodes are managed.

By using Keepalived, you make sure that the services running on Docker Swarm remain accessible at all times, providing an extra layer of high availability on top of the replication and scaling features of Docker Swarm.

Simple Explanation of a 3-Node Docker Swarm with Keepalived

In a 3-node Docker Swarm setup, you have three Raspberry Pis (or servers) working together to run your services. Docker Swarm distributes your services across the nodes for high availability and load balancing.

By adding Keepalived to this setup, you can ensure that the cluster is always accessible through a single virtual IP (VIP). In this example, the VIP is `192.168.0.200`. Keepalived monitors the health of the nodes, and if the active node fails, it automatically assigns the VIP to another healthy node.

So, any traffic to `192.168.0.200` is always directed to an available node, ensuring uninterrupted service even if one node goes down.

Step-by-Step Guide: Setup Keepalived



Keepalived on Docker Swarm A Custom Raspberry Pi Setup

Keepalived is an open-source tool used to ensure high availability and redundancy for services. It does this by monitoring the health of your network and services and automatically switching to a backup server if the primary one fails. In this setup, we are using a custom Keepalived Docker image built specifically for Raspberry Pi (ARMv8) by Takabu, a friend of mine. This allows us to implement Keepalived in a Docker Swarm environment on Raspberry Pis.

In our Docker Swarm setup, Keepalived is used to provide a Virtual IP (VIP) that will always point to a healthy node, ensuring seamless failover if a node goes offline.

Priority List and How it Works

Keepalived assigns a priority to each node. The node with the **highest priority** becomes the primary holder of the VIP. If that node goes down, the node with the **next highest priority** takes over the VIP. This ensures that the VIP is always assigned to a functioning node.

Here's how the priority system works in our Docker Swarm:

- **Swarm1 (192.168.0.10):** Priority `100` (lowest priority)
- **Swarm2 (192.168.0.11):** Priority `101`
- **Swarm3 (192.168.0.12):** Priority `102` (highest priority)

The node with priority 102 (`Swarm3`) will be the primary node, and if it goes down, the VIP will switch to the node with priority 101 (`Swarm2`), and so on.

Setting Up the Priority List

Before deploying Keepalived, you need to set the priority for each node. Use the following commands to label each node with its priority:

```
docker node ls # List all nodes in the swarm

# Assign priority labels to each node:
docker node update Swarm1 --label-add KEEPALIVED_PRIORITY=100
docker node update Swarm2 --label-add KEEPALIVED_PRIORITY=101
docker node update Swarm3 --label-add KEEPALIVED_PRIORITY=102
```

With the priorities set, Keepalived will ensure that the node with the highest priority is always assigned the VIP. If that node fails, Keepalived automatically shifts the VIP to the next highest-priority node, maintaining service availability.

Docker Compose File for Keepalived

```
version: '3.8'

services:
  keepalived:
    image: takabu/public:docker-swarm-keepalived # Custom Keepalived image for Raspberry Pi (ARMv8)
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock # Mount Docker socket to interact with Docker API and control the nodes
```

networks:

- host # Use the host network for direct access to networking features

deploy:

mode: global # Ensures Keepalived runs on all manager nodes

placement:

constraints: [node.role == manager] # Limit deployment to Swarm manager nodes only

environment:

KEEPALIVED_VIRTUAL_IPS: "192.168.0.200" # Virtual IP (VIP) for Keepalived to manage and switch between nodes

networks:

host:

external: true

name: host # Leverage the host network to manage VIP switching and network traffic directly

Explanation of Key Sections:

- **/var/run/docker.sock:/var/run/docker.sock**: This mounts the Docker socket into the Keepalived container, allowing Keepalived to communicate with Docker and control the failover between nodes based on the priority list.
- **KEEPALIVED_VIRTUAL_IPS: "192.168.0.200"**: This environment variable specifies the VIP (192.168.0.200) that Keepalived will manage. This IP will always point to the highest-priority, healthy node.