

# Lab 6 - GlusterFS

BI12-149 - Kieu Huy Hai

April 2024

## 1 System Architecture

The system consists of 2 nodes in a trusted pool, which will create a distributed replicated volume exposed through GlusterFS:

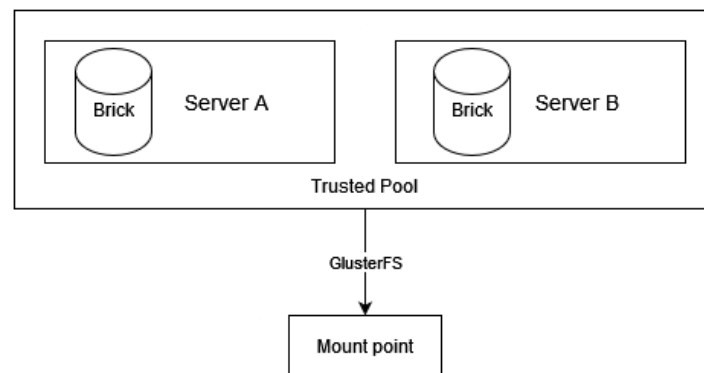


Figure 1: System Architecture

## 2 Setup GlusterFS and Trusted Pool

### 2.1 GlusterFS

We can easily install GlusterFS using the official installation guide:

---

# Download the GPG key

```
wget -O -
    https://download.gluster.org/pub/gluster/glusterfs/9/rsa.pub |
    gpg --dearmor > /etc/apt/trusted.gpg.d/gluster.gpg
# Add the source
DEBID=$(grep 'VERSION_ID=' /etc/os-release | cut -d '=' -f 2 | tr
    -d ' "')
DEBVER=$(grep 'VERSION=' /etc/os-release | grep -Eo '[a-z]+')
DEBARCH=$(dpkg --print-architecture)
echo "deb [signed-by=/etc/apt/trusted.gpg.d/gluster.gpg]
    https://download.gluster.or"
# Install
apt-get update && apt-get install glusterfs-server -y
```

---

## 2.2 Trusted Pool

First, we need to configure the firewall to allow connection between 2 nodes:

```
iptables -I INPUT -p all -s IP_addr -j ACCEPT
```

---

Next, we need to edit `/etc/hosts/` file so that 2 nodes can resolve hostname of each other. After that, we can connect those 2 nodes:

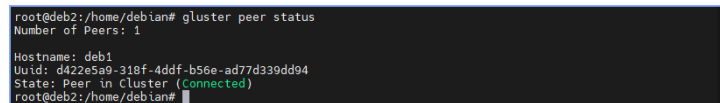
```
gluster peer probe node
```

---

The connection result can be checked using:

```
gluster peer status
```

---



```
root@deb2:/home/debian# gluster peer status
Number of Peers: 1
Hostname: deb1
Uuid: d422e5a9-318f-4ddf-b56e-ad77d339dd94
State: Peer in Cluster (Connected)
root@deb2:/home/debian#
```

Figure 2: Trusted Pool setup successfully

## 3 Setup XFS Partition

We will be created a XFS partition inside a disk. This partition will be used as a "brick" for GlusterFS replicated distributed volume.

To create, first run

---

```
fdisk /dev/disk
```

---

with `disk` is the disk in the server (`sda`, `sdb`, ...). Then, select `n` to create a new partition, provide needed information (for empty disk, normally we just need to press `Enter` to skip through all part). Finally, press `w` to save our change and exit.

Next, we need to install `xfsprogs` through `apt` to be able to format our newly created partition into XFS. After installing, we can run

---

```
mkfs.xfs -i size=512 /dev/disk__partition
```

---

to create a XFS partition. After that, we add an entry to `/etc/fstab` to expose our partition:

---

```
echo "/dev/disk /export/disk xfs defaults 0 0" >> /etc/fstab
systemctl daemon-reload
```

---

Finally, mount the partition:

---

```
mkdir -p /export/sdb1 && mount -a
```

---

## 4 Setup Replicated Distributed Volume

After created the same XFS partitions in 2 nodes, we can finally create a replicated distributed volume using:

---

```
gluster volume create vol_name replica 2 transport tcp
node1:/export/sdb1/brick node2:/export/sdb1/brick force
```

---

and start the volume using:

---

```
gluster volume start vol_name
```

---

We can check the newly created volume using

---

```
gluster volume info
```

---

```
root@deb2:/home/debian# gluster volume info

Volume Name: vol
Type: Replicate
Volume ID: 3fdcee74-04ad-45b4-8f1d-6b70b2dc2a3e
Status: Started
Snapshot Count: 0
Number of Bricks: 1 x 2 = 2
Transport-type: tcp
Bricks:
Brick1: deb1:/export/sdb1/brick
Brick2: deb2:/export/sdb1/brick
Options Reconfigured:
cluster.granular-entry-heal: on
storage.fips-mode-rchecksum: on
transport.address-family: inet
nfs.disable: on
performance.client-io-threads: off
```

Figure 3: Volume information

To use the volume, we can mount it to the local filesystem:

---

```
mkdir /mnt/glusterfs
mount -t glusterfs hostname:/volname /mnt/glusterfs
```

---

## 5 Benchmark

To benchmark the volume, we use `iozone`. The command is below:

---

```
iozone -w -c -e -i 2 -+n -C -r 64k -s 100M -t X
```

---

with:

- `-w`: tells `iozone` not to delete any files that it accessed, so that subsequent tests can use them.
- `-c -e`: allow measuring the time it takes for data to reach persistent storage.
- `-+n`: skipping re-read and re-write tests.
- `-i 2`: random read/write test.

- -C: display how much each thread participated in the test.
- -r 64k: record size will be 64kB.
- -s 100M: file size for test will be 100MB.
- -t X: using X threads to test.

Here is the results:

```

root@deb1:/home/debian# iotzone -w -c -e -i 2 -+n -C -r 64k -s 100M -t 4
Iotzone: Performance Test of File I/O
Version $Revision: 3.489 $
Compiled for 64 bit mode.
Build: linux-AMD64

Contributors: William Norcott, Don Capps, Isom Crawford, Kirby Collins
Al Slater, Scott Rhine, Mike Wisner, Ken Goss
Steve Landherr, Brad Smith, Mark Kelly, Dr. Alain CYR,
Randy Dunlap, Mark Montague, Dan Million, Gavin Brebner,
Jean-Marc Zucconi, Jeff Blomberg, Benny Halevy, Dave Boone,
Erik Habbington, Kris Strecker, Walter Wong, Joshua Root,
Fabrice Bacchella, Zhenghua Xue, Qin Li, Darren Sawyer,
Vangel Bojaxhi, Ben England, Vikentsi Lapa,
Alexey Skidanov, Sudhir Kumar.

Run began: Mon Apr 15 22:17:50 2024

Setting no unlink
Include close in write timing
Include fsync in write timing
No retest option selected
Record Size 64 kB
File size set to 102400 kB
Command line used: iotzone -w -c -e -i 2 -+n -C -r 64k -s 100M -t 4
Output is in kBytes/sec
Time Resolution = 0.000001 seconds.
Processor cache size set to 1024 kBytes.
Processor cache line size set to 32 bytes.
File stride size set to 17 * record size.
Throughput test with 4 processes
Each process writes a 102400 kByte file in 64 kByte records

Children see throughput for 4 random readers      = 3724011.31 kB/sec
Parent sees throughput for 4 random readers      = 3504875.42 kB/sec
Min throughput per process                       = 665540.44 kB/sec
Max throughput per process                       = 1278306.62 kB/sec
Avg throughput per process                       = 931002.83 kB/sec
Min xfer                                         = 42240.00 kB
Child[0] xfer count = 65792.00 kB, Throughput = 861875.38 kB/sec
Child[1] xfer count = 74048.00 kB, Throughput = 918288.88 kB/sec
Child[2] xfer count = 42240.00 kB, Throughput = 665540.44 kB/sec
Child[3] xfer count = 102400.00 kB, Throughput = 1278306.62 kB/sec

Children see throughput for 4 random writers      = 535767.33 kB/sec
Parent sees throughput for 4 random writers      = 246048.96 kB/sec
Min throughput per process                       = 67341.80 kB/sec
Max throughput per process                       = 288861.62 kB/sec
Avg throughput per process                       = 133941.83 kB/sec
Min xfer                                         = 29120.00 kB
Child[0] xfer count = 36480.00 kB, Throughput = 78480.33 kB/sec
Child[1] xfer count = 51456.00 kB, Throughput = 101083.58 kB/sec
Child[2] xfer count = 29120.00 kB, Throughput = 67341.80 kB/sec
Child[3] xfer count = 102400.00 kB, Throughput = 288861.62 kB/sec

```

Figure 4: Using 4 threads

```

root@deb1:/home/debian# iotest -w -c -e -i 2 -+n -C -r 64k -s 100M -t 1
Iotest: Performance Test of File I/O
Version $Revision: 3.489 $
Compiled for 64 bit mode.
Build: linux-AMD64

Contributors: William Norcott, Don Capps, Isom Crawford, Kirby Collins
Al Slater, Scott Rhine, Mike Wisner, Ken Goss
Steve Landherr, Brad Smith, Mark Kelly, Dr. Alain CYR,
Randy Dunlap, Mark Montague, Dan Million, Gavin Brebner,
Jean-Marc Zucconi, Jeff Blomberg, Benny Halevy, Dave Boone,
Erik Habbinga, Kris Strecker, Walter Wong, Joshua Root,
Fabrice Bacchella, Zhenghua Xue, Qin Li, Darren Sawyer,
Vangel Bojaxhi, Ben England, Vikentsi Lapa,
Alexey Skidanov, Sudhir Kumar.

Run began: Mon Apr 15 22:18:18 2024

Setting no_unlink
Include close in write timing
Include fsync in write timing
No retest option selected
Record Size 64 kB
File size set to 102400 kB
Command line used: iotest -w -c -e -i 2 -+n -C -r 64k -s 100M -t 1
Output is in kBytes/sec
Time Resolution = 0.000001 seconds.
Processor cache size set to 1024 kBytes.
Processor cache line size set to 32 bytes.
File stride size set to 17 * record size.
Throughput test with 1 process
Each process writes a 102400 kByte file in 64 kByte records

Children see throughput for 1 random readers      = 1460874.12 kB/sec
Parent sees throughput for 1 random readers      = 1444984.27 kB/sec
Min throughput per process                       = 1460874.12 kB/sec
Max throughput per process                       = 1460874.12 kB/sec
Avg throughput per process                       = 1460874.12 kB/sec
Min xfer                                          = 102400.00 kB
Child[0] xfer count = 102400.00 kB, Throughput = 1460874.12 kB/sec

Children see throughput for 1 random writers      = 322399.78 kB/sec
Parent sees throughput for 1 random writers      = 321306.24 kB/sec
Min throughput per process                       = 322399.78 kB/sec
Max throughput per process                       = 322399.78 kB/sec
Avg throughput per process                       = 322399.78 kB/sec
Min xfer                                          = 102400.00 kB
Child[0] xfer count = 102400.00 kB, Throughput = 322399.78 kB/sec

```

Figure 5: Using 1 thread

We can conclude that when using more threads, the average read/write throughput become smaller. Also, the reading speed of GlusterFS is much faster than the writing speed.