1.2.2. 一些有趣的 volume

1.2.2.1. 复制卷

官方支持复制卷和分布式复制卷，而生产环境中常用的是分布式复制卷，首先这里引用一些官方的图片，来理解一下什么是分布式复制卷。

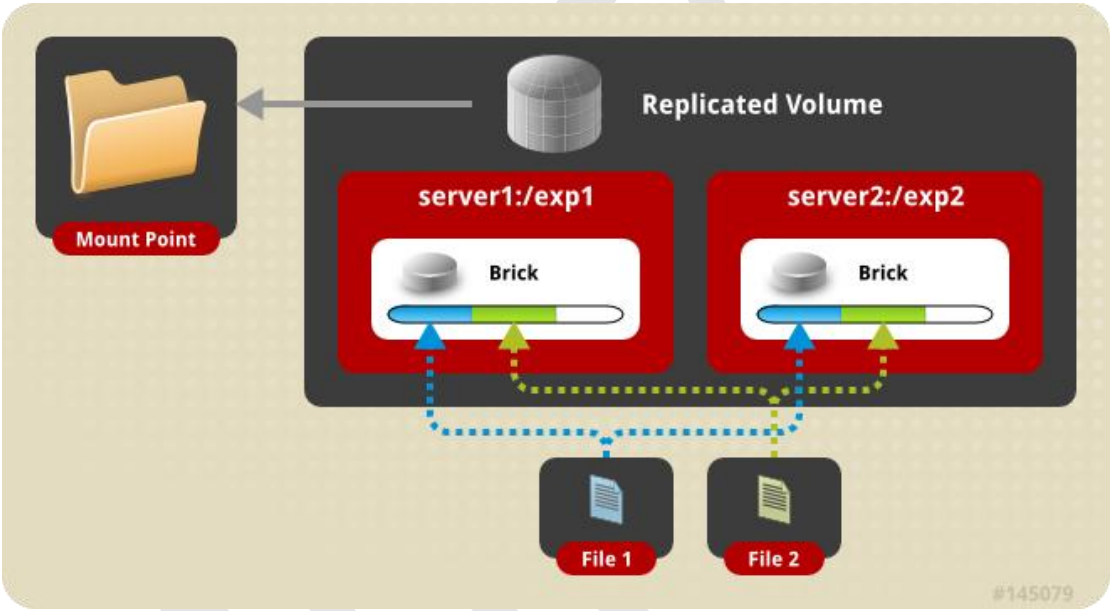


图 1.1

这里图片给出的是一个 2 个副本的复制卷，生产环境中通常使用 3 个副本的复制卷，创建的命令如下所示.

```
1.  [root@gfs01 ~]#  gluster volume create test-replica replica 3 192.16
    8.0.{110,111,112}:/glusterfs/test-replica
2.  volume create: test-replica: failed: The brick 192.168.0.110:/glusterfs
    /test-replica is being created in the root partition. It is recommended th
    at you don't use the system's root partition for storage backend. Or us
    e 'force' at the end of the command if you want to override this behav
    ior.
3.  [root@gfs01 ~]#  gluster volume create test-replica replica 3 192.16
    8.0.{110,111,112}:/glusterfs/test-replica force
```

```
volume create: test-replica: success: please start the volume to access data
[root@gfs01 ~]# gluster volume start test-replica
volume start: test-replica: success
```

这里因为 glusterfs 官方推荐不使用 root 进行操作的，或者使用 force 命令强制创建，那么这里作为测试使用可以不用考虑，但生产环境安装部署的时候需要注意该问题。这里使用命令创建了一个名为 test-replica 的三副本的分布式复制卷，那么这里创建之后并且启动了。那么该如何知道该卷的状态是否正常呢？可以查看状态和挂载使用。

```
# gluster volume status   test-replica
Status of volume: test-replica
Gluster process                                 TCP Port  RDMA Port  Online  Pid
------------------------------------------------------------------------------
Brick 192.168.0.110:/glusterfs/test-replica 49153     0          Y       16485
Brick 192.168.0.111:/glusterfs/test-replica 49152     0          Y       16368
Brick 192.168.0.112:/glusterfs/test-replica 49153     0          Y       16422
Self-heal Daemon on localhost               N/A       N/A        Y       16502
Self-heal Daemon on gfs03                   N/A       N/A        Y       16439
Self-heal Daemon on gfs02                   N/A       N/A        Y       16385

Task Status of Volume test-replica
------------------------------------------------------------------------------
There are no active volume tasks
```

这里重点需要关注两个地方，分别是 online 和 pid，这里如果每个节点上关于该 volume 的存储目录管理进程异常，也就是 brick 异常的话，那么这里就无法获取到对应的端口号的。那么这里还可以使用命令查看一下每个节点的 brick 进程，结果如下所示。

```
# ps -ef |grep 16485
root     16485     1  0 12:27 ?        00:00:00 /usr/sbin/glusterfsd -s 192.168.0.110 --volfile-id test-replica.192.168.0.110.glusterfs-test-replica -p /var/run/gluster/vols/test-replica/192.168.0.110-glusterfs-test-replica.pid -S /var/run/gluster/d403ef3161bd809c.socket --brick-name /glusterfs/test-replica -l /var/log/glusterfs/bricks/glusterfs-test-replica.log --xlator-option *-posix.glusterd-uuid=99827066-72c7-484f-b1e8-0a9c4bc46b54 --process-name brick --brick-port 49153 --xlator-option test-replica-server.listen-port=49153
```

这一段信息中,指定的日志路径默认都是在/var/log/glusterfs 下的,而 brick 的日志则在/var/log/glusterfs/bricks 中，同时如果生产环境中遇到某个 brick 的进程异常，遇到需要手动启动 brick 进程的时候，可以使用上面的输出结果来手动执行，当然一般也很少会遇到这样的情况。

这里想先强调一下，对于复制卷，不建议使用 2 副本的复制，因为这样会在 brick 异常之后,比较容易出现脑裂的现象，关于这部分的内容，会在后面详细讲解一下。

那么这里该使用呢?可以在其他节点上,只要安装了 glusterfs client 的话，那直接使用 mount 挂载即可。

```
[root@gfs03 ~]# gluster volume info test-replica

Volume Name: test-replica
Type: Replicate
Volume ID: 66fed900-29f5-40f5-8add-7e4e365ab29a
Status: Started
Snapshot Count: 0
Number of Bricks: 1 x 3 = 3
Transport-type: tcp
Bricks:
Brick1: 192.168.0.110:/glusterfs/test-replica
```

```
Brick2: 192.168.0.111:/glusterfs/test-replica
Brick3: 192.168.0.112:/glusterfs/test-replica
Options Reconfigured:
performance.client-io-threads: off
nfs.disable: on
transport.address-family: inet
storage.fips-mode-rchecksum: on
cluster.granular-entry-heal: on


[root@gfs03 ~]# mount -t glusterfs 192.168.0.110:test-replica /mnt/test-replica
[root@gfs03 ~]# ls /mnt/test-replica/
[root@gfs03 ~]# touch /mnt/test-replica/a.txt
[root@gfs03 ~]# date >> /mnt/test-replica/a.txt
...
[root@gfs03 ~]# md5sum  /mnt/test-replica/a.txt
9eb6c6683f293a1f62d38a4ed94b17c8  /mnt/test-replica/a.txt
[root@gfs03 ~]# md5sum /glusterfs/test-replica/a.txt
9eb6c6683f293a1f62d38a4ed94b17c8  /glusterfs/test-replica/a.txt


[root@gfs02 ~]#  md5sum /glusterfs/test-replica/a.txt
9eb6c6683f293a1f62d38a4ed94b17c8  /glusterfs/test-replica/a.txt

[root@gfs01 ~]#  md5sum /glusterfs/test-replica/a.txt
9eb6c6683f293a1f62d38a4ed94b17c8  /glusterfs/test-replica/a.txt
```

这里选择的 ip 节点可以是集群中的任意一个节点，哪怕 brick 并不在该节点上面，而这也正是 glusterfs 无中心架构的一个特点.因为通信方式采用的是全互联的模式,因此元数据信息保持一致的情况下，那么这里是可以选择任意节点进行挂载的。另外这里可以看到，复制卷的特点就是数据都是完整的一份的，然而冗余卷则是不一样的，可以进行对比一下.

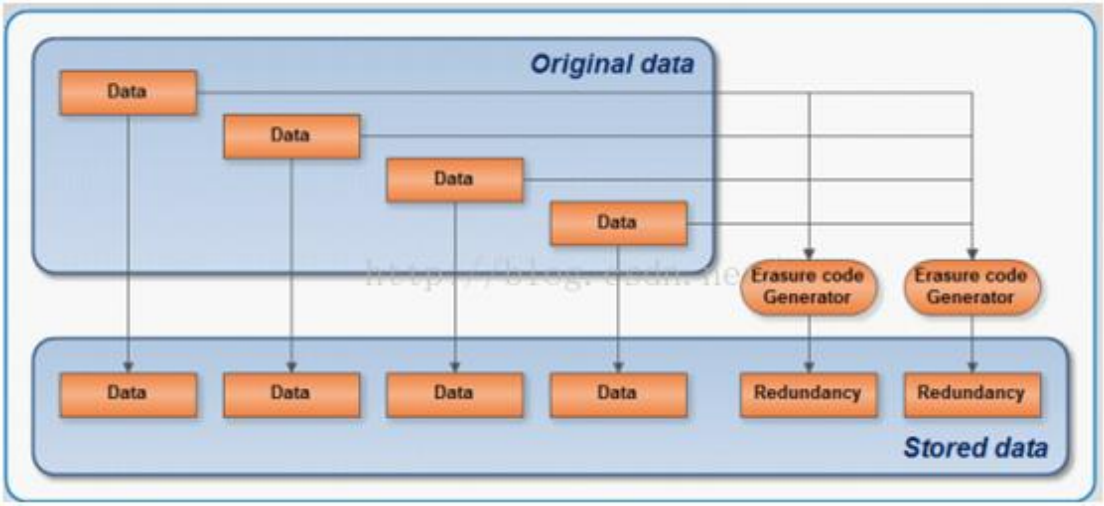
当然在生产环境中，也不建议把挂载的 ip 都集中在其中的一个节点上， 否则所有的请求流量都会经过该节点，会加重节点的负载。而挂载以后，这里的使

用就和本地目录类似了，也和 NFS 的使用类似。最后如果想解除挂载，那么直接使用 umount 目录即可。

```
1.    [root@gfs03 ~]# umount  /mnt/test-replica/
```

1.2.2.2. 冗余卷

什么是冗余卷，这里与复制卷不一样的地方就是，数据到底以怎样的形式存放在节点上，像常见的 hdfs 是 3 副本机制的。而随着数据量越来越多，如果未来对于所有的数据都是 3 副本，那么占用的空间就会比较多了，因此为了节约空间，就有了冗余码的出现，通过下面的图来感受一下。



从图中可以很直观地感受到，所谓的冗余码，就是把一个完整的数据切割成多份，然后每一份计算得到一部分冗余码，并且存储在节点中，这里的存储空间会比完成的一份数据多一些，但是会远远比 3 副本所占用的空间小。同时因为有了冗余码，因此即使在损坏一定的比例数据下，数据的完整性都能得到保证。

那么这里的数据是如何存储地，冗余码如何计算得到的？通过计算得到这些冗余码等问题，这些将放在后面详细讲解，因为这部分内容涉及到比较多的数学内容，如果感兴趣，也可以先自行简单了解一下里德-所罗门码(Reed-solomon codes)。同时目前工业界也比较关注，未来到底是复制卷还是冗余卷，该如何使用，优缺点如何，都是非常值得思考与关注的问题。

这里我们先了解一下如何创建分布式冗余卷，并且这里我们挂载创建一个文件，写入一些数据，查看一下到底分布式冗余卷的数据是如何分布的。

```
[root@gfs01 ~]# gluster volume create test-disperse disperse 3 192.168.0.{110,111,112}:/glusterfs/test-disperse  force
  volume create: test-disperse: success: please start the volume to access data

[root@gfs01 ~]# gluster volume start  test-disperse
volume start: test-disperse: success

[root@gfs01 ~]# gluster volume info test-disperse

Volume Name: test-disperse
Type: Disperse
Volume ID: 0390d729-b6d8-4edd-bb72-bd28c3ec7472
Status: Started
Snapshot Count: 0
Number of Bricks: 1 x (2 + 1) = 3
Transport-type: tcp
Bricks:
Brick1: 192.168.0.110:/glusterfs/test-disperse
Brick2: 192.168.0.111:/glusterfs/test-disperse
Brick3: 192.168.0.112:/glusterfs/test-disperse
Options Reconfigured:
storage.fips-mode-rchecksum: on
transport.address-family: inet
nfs.disable: on


[root@gfs01 ~]# mount -t glusterfs 192.168.0.112:test-disperse /mnt/test-disperse
[root@gfs01 ~]# touch /mnt/test-disperse/1.txt
[root@gfs01 ~]# date >> /mnt/test-disperse/1.txt
```

```
[root@gfs01 ~]# date >> /mnt/test-disperse/1.txt
...
[root@gfs01 ~]# cat /mnt/test-disperse/1.txt
Wed May 26 11:42:49 EDT 2021
Wed May 26 11:42:49 EDT 2021
Wed May 26 11:42:50 EDT 2021
Wed May 26 11:42:51 EDT 2021
Wed May 26 11:42:51 EDT 2021

[root@gfs01 ~]# md5sum  /mnt/test-disperse/1.txt
95af6ad88c70c127faee8d84e3eb8d8f  /mnt/test-disperse/1.txt
[root@gfs01 ~]# md5sum  /glusterfs/test-disperse/1.txt
0215b19719d86fd12a973926ec0c0854   /glusterfs/test-disperse/1.txt


[root@gfs02 ~]#  md5sum  /glusterfs/test-disperse/1.txt
cdcf65a53fef2a8066bd96d6324dab85  /glusterfs/test-disperse/1.txt

[root@gfs03 ~]#  md5sum  /glusterfs/test-disperse/1.txt
d76f9fc89d264d19641ec552f600d46a  /glusterfs/test-disperse/1.txt
```

我们查看一下每一个 brick 下面的数据情况可以发现文件的 md5 都不一样的，那么这里并不是和完整的挂载目录文件，同时如果感兴趣，这里可以查看一下几个 brick 下面的数据分布情况。

1.2.2.3. force 的作用

前面提到了两种常见的 volume,那么对于 volume 的操作有 status,start 和 info 这些常见的，而其中对于 start 和 stop 都有一个可选项 force，这个的作用其实是什么呢？这里我们可以简单测试一下。

```
[root@gfs01 ~]# gluster volume start

Usage:
volume start <VOLNAME> [force]
```

```
5.
6.  [root@gfs01 ~]# gluster volume stop
7.
8.  Usage:
9.  volume stop <VOLNAME> [force]
```

下面进行简单的测试，方法就是 kill 掉其中一个 brick 的目录，然后重新启动查看一下作用。

```
1.  Status of volume: test-replica
2.  Gluster process                                 TCP Port  RDMA Port  Online  Pid
3.  ------------------------------------------------------------------------------
4.  Brick 192.168.0.110:/glusterfs/test-replica 49153     0          Y       1776
5.  Brick 192.168.0.111:/glusterfs/test-replica 49153     0          Y       1620
6.  Brick 192.168.0.112:/glusterfs/test-replica 49155     0          Y       1521
7.  Self-heal Daemon on localhost               N/A       N/A        Y       1160
8.  Self-heal Daemon on gfs02                   N/A       N/A        Y       1426
9.  Self-heal Daemon on gfs03                   N/A       N/A        N       N/A
10.
11. Task Status of Volume test-replica
12. ------------------------------------------------------------------------------
13. There are no active volume tasks
14.
15. //这里删掉了其中一个 brick 目录
16. [root@gfs01 ~]# rm -fr /glusterfs/test-replica
17. [root@gfs01 ~]# gluster volume start  test-replica
18. volume start: test-replica: failed: Failed to find brick directory /glusterfs/test-replica for volume test-replica. Reason : No such file or directory
19.
20. //前面无法正常启动了.使用 force 强制启动.
21. [root@gfs01 ~]# gluster volume start  test-replica  force
22. volume start: test-replica: success
23. [root@gfs01 ~]# gluster volume status     test-replica
24. Status of volume: test-replica
25. Gluster process                                 TCP Port  RDMA Port  Online  Pid
```

```
------------------------------------------------------------------------------
Brick 192.168.0.110:/glusterfs/test-replica N/A       N/A        N       N/A

Brick 192.168.0.111:/glusterfs/test-replica 49154     0          Y       167
4
Brick 192.168.0.112:/glusterfs/test-replica 49154     0          Y       157
3
Self-heal Daemon on localhost               N/A       N/A        Y       1160
Self-heal Daemon on gfs03                   N/A       N/A        N       N/A
Self-heal Daemon on gfs02                   N/A       N/A        Y       1426

Task Status of Volume test-replica
------------------------------------------------------------------------------
There are no active volume tasks
```

从这里可以看到，其中所谓的 force 就是在一些 brick 坏掉异常的时候，能够强制启动，对于 3 副本的分布式复制卷来说，正常是只要超过 2 个 brick 正常就可以启动的，但是默认是不行，需要使用 force 命令。另外这里如果感兴趣，读者可以自行测试一下，前面删掉的数据目录，如果再次重新创建，能否继续生效呢？ 另外这里 force 的作用可以从代码中找到答案。

```
// xlator->mgmt->glusterd->src->glusterd-volume-ops.c
//函数名: glusterd_op_stage_start_volume

ret = gf_lstat_dir(brickinfo->path, NULL);
        if (ret && (flags & GF_CLI_FLAG_OP_FORCE)) {
            continue;
        } else if (ret) {
            len = snprintf(msg, sizeof(msg),
                           "Failed to find "
                           "brick directory %s for volume %s. "
                           "Reason : %s",
                           brickinfo->path, volname, strerror(errno));
            if (len < 0) {
                strcpy(msg, "<error>");
            }
            goto out;
        }
```