DRBD 8.4 and GFS2 with Pacemaker

Matt Kereczman
Version 1.1, 2017-04-24

Table of Contents

1. Introduction

This guide will walk you through configuring a dual-primary DRBD device as the backing storage for a GFS2 clustered filesystem, all managed by the Pacemaker cluster resource manager.

The end result in this guide will be a simple log file being written to simultaneously by two nodes; not very impressive. However, the design would be the same for something more impressive, like a virtual machine capable of live migrations, or a highly available web server with load balancing across all nodes.

2. Assumptions

This guide assumes the following system configurations:

2.1. System Configurations

Hostname	DRBD Backing Device	Management Interface	Management IP	Crossover Interface	Crossover IP
node-a	/dev/sdb	eth0	192.168.10.75	eth1	172.16.0.75
node-b	/dev/sdb	eth0	192.168.10.76	eth1	172.16.0.76

Hostnames should be resolvable between nodes. TIP: Optional: configuring passwordless ssh key-based authentication for root between nodes is useful for utilities included in `crmsh`, but are not discussed in this guide.

2.2. Firewall Configuration

Refer to your firewall documentation for how to open/allow ports. You will need the following ports open in order for your cluster to function properly.

Ports

Component	Protocol	Port
DRBD	TCP	7788
Corosync	UDP	5404, 5405
GFS2	TCP	2224, 3121, 21064

2.3. SELinux

If you have SELinux enabled, you must either put it into `permissive` mode, or disable it completely. This is required by GFS2.

3. Register Nodes with LINBIT

We will install DRBD from LINBIT's repositories. To access those repositories you will need to have been setup in LINBIT's system, and have access to the LINBIT customer portal.

Once you have access to the customer portal, you can register and configure your node's repository access by using the Python command line tool outlined in the "REGISTER NODES" section of the portal.

To register the cluster nodes and configure LINBIT's repositories, run the following on all nodes, one at a time:

```
# curl -O https://my.linbit.com/linbit-manage-node.py
# chmod +x ./linbit-manage-node.py
# ./linbit-manage-node.py
```

The script will prompt you for your LINBIT portal username and password. Once provided it will list cluster nodes associated with your account (none at first).

After you tell the script which cluster to register the node with, you will be asked a series of questions regarding which repositories you'd like to enable; we will need **`pacemaker-1.1`** and **`drbd-8.4`** enabled for this guide.

Be sure to say **yes** to the questions regarding installing LINBIT's public key to your keyring and writing the repository configuration file.

After that, you should be able to run, `# yum info kmod-drbd`, and see that yum is pulling package information from LINBIT's repositories.

Before installing packages, we want to make sure we only pull the cluster stack packages from LINBIT's repositories.

We will need to add an exclude line to the `[base]` and `[updates]` section of `/etc/yum.repos.d/CentOS-Base.repo`. The modified `[base]` and `[updates]` section should look like this:

```
[base]
name=CentOS-$releasever - Base
mirrorlist=http://mirrorlist.centos.org/?release=$releasever&arch=$ba[...]
#baseurl=http://mirror.centos.org/centos/$releasever/os/$basearch/
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-7
exclude=pacemaker* corosync* cluster* drbd* resource-agents

#released updates
[updates]
name=CentOS-$releasever - Updates
mirrorlist=http://mirrorlist.centos.org/?release=$releasever&arch=$ba[...]
#baseurl=http://mirror.centos.org/centos/$releasever/updates/$basearch/
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-7
exclude=pacemaker* corosync* cluster* drbd* resource-agents
```

4. Install Cluster Stack Packages

Install The cluster stack packages and dependencies.

```
# yum install drbd kmod-drbd linbit-cluster-stack-corosync2 gfs2-utils lvm2-cluster fence-agents-all
```

5. Configure Cluster Stack

With the cluster stack packages installed, we can begin configuring the base of the cluster.

5.1. DRBD

We'll configure our DRBD resource, `r0`, with the required settings for running in dual-primary mode.

```
resource r0 {
    protocol C;
    device /dev/drbd0;
    meta-disk internal;
    disk /dev/sdb;
    net {
        allow-two-primaries;
    }
    disk {
        fencing resource-and-stonith;
    }
    handlers {
        fence-peer "/usr/lib/drbd/crm-fence-peer.sh";
        after-resync-target "/usr/lib/drbd/crm-unfence-peer.sh";
    }
    on node-a {
        address 172.16.0.75:7788;
    }
    on node-b {
        address 172.16.0.76:7788;
    }
}
```

Since we're going to be controlling DRBD through Pacemaker, we'll want to disable systemd from starting DRBD at boot on both nodes.

We'll also need to create the DRBD device's metadata and bring the device up on both nodes.

```
# systemctl disable drbd
# drbdadm create-md r0
# drbdadm up r0
```

We can skip the initial sync by generating a new current UUID; only perform this step on one node. Then, you should see you have a `Connected` and `UpToDate/UpToDate` device.

```
# drbdadm new-current-uuid --clear-bitmap r0/0
# cat /proc/drbd
version: 8.4.9-1 (api:1/proto:86-101)
GIT-hash: 9976da086367a2476503ef7f6b13d4567327a280 build by buildsystem@linbit, 2016-11-03
21:50:42
 0: cs:Connected ro:Secondary/Secondary ds:UpToDate/UpToDate C r-----
    ns:0 nr:0 dw:0 dr:0 al:16 bm:0 lo:0 pe:0 ua:0 ap:0 ep:1 wo:f oos:0
```

5.2. Corosync

We'll configure Corosync to use both our interfaces, making sure to use the replication/crossover link as the primary interface, `ring0` in Corosync terminology, communications channel.

```
totem {
  version: 2
  secauth: off
  cluster_name: drbd-gfs2
  transport: udpu
  rrp_mode: passive
}

nodelist {
  node {
    ring0_addr: 172.16.0.75
    ring1_addr: 192.168.10.75
    nodeid: 1
  }
  node {
    ring0_addr: 172.16.0.76
    ring1_addr: 192.168.10.76
    nodeid: 2
  }
}

quorum {
  provider: corosync_votequorum
  two_node: 1
}

logging {
  to_syslog: yes
}
```

Note the `cluster_name`; you will need it when creating the GFS2 filesystem later in this guide.

We usually want Corosync to start at boot in order for a node to rejoin the cluster after a reboot; so we'll set it to enabled and start it so we can begin configuring Pacemaker:

```
# systemctl enable corosync
# systemctl start corosync
```

5.3. Pacemaker

Since we set Corosync to start at boot, we also need Pacemaker starting at boot; enable and start pacemaker so we can begin configuring our cluster properties:

```
# systemctl enable pacemaker
# systemctl start pacemaker
```

The cluster stack is now started, and we can configure our cluster properties.

Pacemaker defaults to `no-quorum-policy=stop`, but for GFS2, we cannot stop the filesystem without having quorum; so that won't work. Instead, we'll tell the cluster stack to `freeze` without quorum, meaning the remaining node will take no cluster actions until quorum is regained.

```
# crm configure
crm(live)configure# property no-quorum-policy=freeze
```

5.3.1. DLM

The distributed lock manager (DLM) is required by clvmd and GFS2 for access control and file locking. Add DLM to that cluster, and clone it so it runs on all nodes:

```
crm(live)configure# primitive dlm ocf:pacemaker:controld \
   > op start interval=0s timeout=90s \
   > op stop interval=0s timeout=100s \
   > op monitor interval=30s on-fail=fence
crm(live)configure# clone dlm-clone dlm meta interleave=true ordered=true
```

5.3.2. STONITH

You should always use fencing when possible in a Pacemaker cluster. This is a hard requirement when using a clustered filesystem like GFS2.

```
crm(live)configure# primitive st_node-a stonith:external/ipmi \
   > params hostname=node-a userid=root passwd=calvin ipaddr=192.168.0.120 interface=lanplus \
   > op start interval=0s timeout=60s \
   > op stop interval=0s timeout=60s \
   > op monitor interval=3600s timeout=60s
crm(live)configure# primitive st_node-b stonith:external/ipmi \
   > params hostname=node-b userid=root passwd=calvin ipaddr=192.168.0.120 interface=lanplus \
   > op start interval=0s timeout=60s \
   > op stop interval=0s timeout=60s \
   > op monitor interval=3600s timeout=60s
crm(live)configure# location l_st-node-a_never-on_node-a st_node-a -inf: node-a
crm(live)configure# location l_st-node-b_never-on_node-b st_node-b -inf: node-b
crm(live)configure# verify
crm(live)configure# commit
crm(live)configure# exit

# crm_mon -1D
Online: [ node-b node-a ]

 st_node-a      (stonith:external/ipmi):        Started node-b
 st_node-b      (stonith:external/ipmi):        Started node-a
 Clone Set: dlm-clone [dlm]
     Started: [ node-b node-a ]
```

STONITH device configurations will vary depending on hardware.

WARN: Again, STONITH is a hard requirement for DLM; you will not be able to proceed without some form of STONITH.

Now that the cluster stack is prepared, and the DLM is running on both nodes, we can begin configuring the clustered storage.

6. CLVM

We'll begin preparing our storage by configuring clvmd. We'll enable cluster locking in the LVM configuration on both nodes.

```
# /sbin/lvmconf --enable-cluster
```

Then, from a single node, we'll add a cloned clvmd resource to the cluster configuration.

```
# crm configure
crm(live)configure# primitive clvmd ocf:heartbeat:clvm \
   > op start interval=0s timeout=90s \
   > op stop interval=0s timeout=90s \
   > op monitor interval=30s timeout=90s on-fail=fence
crm(live)configure# clone clvmd-clone clvmd \
   > meta interleave=true ordered=true
crm(live)configure# colocation cl_clvmd-with-dlm inf: clvmd-clone dlm-clone
crm(live)configure# order o_dlm-before-clvmd inf: dlm-clone:start clvmd-clone:start
crm(live)configure# verify
crm(live)configure# commit
```

You'll see warnings when you scan after enabling cluster lvm locking. This is because the `lvmconf` command issued above sets a configuration option that disables the `lvm2-lvmetad` service from starting at boot, but doesn't stop it. If you want to avoid these warnings, you can issue the following command (or reboot) on both nodes: `systemctl stop lvm2-lvmetad.socket lvm2-lvmetad.service`

We are going to put our clvm for GFS2 on top of our DRBD device. To do so, we need to tell LVM to ignore DRBD's backing disk when scanning for LVM signatures; making it so LVM doesn't touch DRBD's backing disk when scanning devices.

First, disable the writing of LVM cache and remove any existing caches:

```
# sed -i 's/write_cache_state = 1/write_cache_state = 0/' /etc/lvm/lvm.conf
# rm /etc/lvm/cache/*
```

Edit the `filter` option in `/etc/lvm/lvm.conf` to reject `/dev/sdb`:

```
    # This configuration option has an automatic default value.
    # filter = [ "a|.*/|" ]
    filter = [ "r|^/dev/sdb$|" ]
```

We can test our filter by creating the clustered LVM on our DRBD device, and then scanning to ensure we're not seeing a duplicate signature on `/dev/sdb`, DRBD's backing disk, and `/dev/drbd0`.

Once the filter is correct, promote DRBD to Primary on both nodes:

```
# drbdadm primary r0
# cat /proc/drbd
version: 8.4.9-1 (api:1/proto:86-101)
GIT-hash: 9976da086367a2476503ef7f6b13d4567327a280 build by buildsystem@linbit, 2016-11-03
21:50:42
 0: cs:Connected ro:Primary/Primary ds:UpToDate/UpToDate C r-----
    ns:0 nr:0 dw:0 dr:912 al:0 bm:0 lo:0 pe:0 ua:0 ap:0 ep:1 wo:f oos:0
```

Then, from either node (not both), create the clustered volume group and logical volume:

```
# vgcreate -Ay -cy cl_vg /dev/drbd0
  Physical volume "/dev/drbd0" successfully created.
  Clustered volume group "cl_vg" successfully created
# lvcreate -L500G -n r0 cl_vg
  Logical volume "r0" created.
```

Finally, check our clustered LVM and be sure to heed any warnings:

```
# pvs
  PV         VG               Fmt  Attr PSize   PFree
  /dev/drbd0 cl_vg            lvm2 a--  930.47g 430.47g
  /dev/sda2  centos_fellowship lvm2 a--  110.76g  64.00m
# vgdisplay cl_vg
  --- Volume group ---
  VG Name               cl_vg
  System ID
  Format                lvm2
  Metadata Areas        1
  Metadata Sequence No  2
  VG Access             read/write
  VG Status             resizable
  Clustered             yes
  Shared                no
  MAX LV                0
  Cur LV                1
  Open LV               1
  Max PV                0
  Cur PV                1
  Act PV                1
  VG Size               930.47 GiB
  PE Size               4.00 MiB
  Total PE              238200
  Alloc PE / Size       128000 / 500.00 GiB
  Free  PE / Size       110200 / 430.47 GiB
  VG UUID               d3GYSL-xGM1-kdVP-6VIf-WrJK-ZX4R-mSkG6p
# lvdisplay cl_vg
  --- Logical volume ---
  LV Path                /dev/cl_vg/r0
  LV Name                r0
  VG Name                cl_vg
  LV UUID                eSDJos-3CL7-0uEY-7aLA-0Az5-FNuy-NEgane
  LV Write Access        read/write
  LV Creation host, time node-a, 2017-03-15 11:11:44 -0700
  LV Status              available
  # open                 0
  LV Size                500.00 GiB
  Current LE             128000
  Segments               1
  Allocation             inherit
  Read ahead sectors     auto
  - currently set to     256
  Block device           253:3
```

If everything looks good, we can move onto creating the GFS2 filesystem.

7. GFS2

Create the GFS2 filesystem using the command below, being certain to create enough journals for each node in the cluster (two if you're following this guide), and that you set the lockspce to `clustername:lockspace`, where clustername is the `cluster_name` we defined in our `corosync.conf`, and `lockspace` (though arbitrary) is the lvm name we're putting our filesystem on:

```
# mkfs.gfs2 -j2 -p lock_dlm -t drbd-gfs2:r0 /dev/cl_vg/r0
/dev/cl_vg/r0 is a symbolic link to /dev/dm-3
This will destroy any data on /dev/dm-3
Are you sure you want to proceed? [y/n]y
Discarding device contents (may take a while on large devices): Done
Adding journals: Done
Building resource groups: Done
Creating quota file: Done
Writing superblock and syncing: Done
Device:                    /dev/cl_vg/r0
Block size:                4096
Device size:               500.00 GB (131072000 blocks)
Filesystem size:           500.00 GB (131071997 blocks)
Journals:                  2
Resource groups:           2001
Locking protocol:          "lock_dlm"
Lock table:                "drbd-gfs2:r0"
UUID:                      82576cb5-c64a-5bc4-ac45-d0738cf5752a
```

8. Add Cluster Services to Pacemaker

Now that we have our filesystem created, we can bring it all together in our Pacemaker configuration.

8.1. Add DRBD to Pacemaker

Add the DRBD device to our Pacemaker configuration. We'll also need to add constraints to be sure each of our devices are promoted before starting DLM on that node.

```
# crm configure
crm(live)configure# primitive p_drbd_r0 ocf:linbit:drbd \
   > params drbd_resource=r0 \
   > op monitor interval=29s role=Master \
   > op monitor interval=31s role=Slave \
   > op start interval=0s timeout=240s \
   > op stop interval=0s timeout=100s
crm(live)configure# ms ms_drbd_r0 p_drbd_r0 \
   > meta master-max=2 master-node-max=1 clone-node-max=1 clone-max=2 notify=true
interleave=true
crm(live)configure# colocation cl_dlm-with-drbd inf: dlm-clone ms_drbd_r0:Master
crm(live)configure# order o_drbd-before-dlm inf: ms_drbd_r0:promote dlm-clone:start
crm(live)configure# verify
crm(live)configure# commit
crm(live)configure# exit
```

You should now see your DRBD device is being managed by the cluster:

```
# crm_mon -1Dr
Online: [ node-b node-a ]

Full list of resources:

 st_node-a      (stonith:external/ipmi):        Started node-b
 st_node-b      (stonith:external/ipmi):        Started node-a
 Clone Set: dlm-clone [dlm]
     Started: [ node-b node-a ]
 Clone Set: clvmd-clone [clvmd]
     Started: [ node-b node-a ]
 Master/Slave Set: ms_drbd_r0 [p_drbd_r0]
     Masters: [ node-b node-a ]
```

8.2. Add GFS2 to Pacemaker

Add the GFS2 filesystem mount to our Pacemaker configuration. Set constraints so the filesystem doesn't start until after the DRBD device is promoted on that node:

```
# crm configure
crm(live)configure# primitive gfs2_r0 ocf:heartbeat:Filesystem \
   > params device="/dev/cl_vg/r0" directory="/mnt/gfs2" fstype=gfs2 options=noatime \
   > op start interval=0s timeout=60s \
   > op stop interval=0s timeout=60s \
   > op monitor interval=10s timeout=40s on-fail=fence
crm(live)configure# clone gfs2_r0-clone gfs2_r0 meta interleave=true
crm(live)configure# colocation cl_gfs2_r0-with-clvmd inf: gfs2_r0-clone clvmd-clone
crm(live)configure# order o_clvmd-before-gfs2_r0 inf: clvmd-clone:start gfs2_r0-clone:start
crm(live)configure# verify
crm(live)configure# commit
crm(live)configure# exit
```

The `Filesystem` resource agent will create the mountpoint if it does not exist.

You should now see your GFS2 filesystem is mounted and managed by Pacemaker:

```
# mount | grep gfs2
/dev/mapper/cl_vg-r0 on /mnt/gfs2 type gfs2 (rw,noatime,seclabel)
# crm_mon -1Dr
Online: [ node-b node-a ]

Full list of resources:

 st_node-a      (stonith:external/ipmi):        Started node-b
 st_node-b      (stonith:external/ipmi):        Started node-a
 Clone Set: dlm-clone [dlm]
     Started: [ node-b node-a ]
 Clone Set: clvmd-clone [clvmd]
     Started: [ node-b node-a ]
 Master/Slave Set: ms_drbd_r0 [p_drbd_r0]
     Masters: [ node-b node-a ]
 Clone Set: gfs2_r0-clone [gfs2_r0]
     Started: [ node-b node-a ]
```

9. GFS2 and Failure Testing

Now that we have everything defined in the cluster, we should verify that it's working properly. I will use the following bash script, to append messages to a text file on the GFS2 device from both nodes.

```
#!/bin/bash
while true; do
    echo "$HOSTNAME : $(date)" >> /mnt/gfs2/log.txt
    sleep 1s
done

# this shouldn't exit, so
exit 1
```

Run that script and `tail` the `log.txt` and see that we're getting messages from both nodes.

```
# ./testing-gfs2.sh &
# tail -F /mnt/gfs2/log.txt
node-a : Wed Mar 15 11:46:37 PDT 2017
node-b : Wed Mar 15 11:46:37 PDT 2017
node-a : Wed Mar 15 11:46:38 PDT 2017
node-b : Wed Mar 15 11:46:39 PDT 2017
node-a : Wed Mar 15 11:46:40 PDT 2017
node-b : Wed Mar 15 11:46:40 PDT 2017
node-a : Wed Mar 15 11:46:41 PDT 2017
node-b : Wed Mar 15 11:46:42 PDT 2017
node-a : Wed Mar 15 11:46:43 PDT 2017
node-b : Wed Mar 15 11:46:43 PDT 2017
node-a : Wed Mar 15 11:46:44 PDT 2017
node-b : Wed Mar 15 11:46:44 PDT 2017
node-a : Wed Mar 15 11:46:45 PDT 2017
node-b : Wed Mar 15 11:46:46 PDT 2017
node-a : Wed Mar 15 11:46:47 PDT 2017
node-b : Wed Mar 15 11:46:47 PDT 2017
node-a : Wed Mar 15 11:46:48 PDT 2017
node-b : Wed Mar 15 11:46:50 PDT 2017
node-a : Wed Mar 15 11:46:50 PDT 2017
node-b : Wed Mar 15 11:46:51 PDT 2017
```

Now forkbomb a node, watch it get fenced, and writes should continue on the surviving node. There are many other ways to test failure scenarios, and I will leave that as an excercise for the reader.

Appendix A: Additional Information and Resources

- The DRBD® User's Guide: http://www.drbd.org/users-guide/
- The DRBD® Mailing Lists: http://www.drbd.org/home/mailinglists/
- We're also on IRC. You can find us on Freenode.net in #drbd.

Appendix B: Legalese

B.1. Trademark Notice

DRBD® and LINBIT® are trademarks or registered trademarks of LINBIT in Austria, the United States, and other countries. Other names mentioned in this document may be trademarks or registered trademarks of their respective owners.

B.2. License Information

The text and illustrations in this document are licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported license ("CC BY-SA").

- A summary of CC BY-NC-SA is available at http://creativecommons.org/licenses/by-nc-sa/3.0/.
- The full license text is available at http://creativecommons.org/licenses/by-nc-sa/3.0/legalcode.
- In accordance with CC BY-NC-SA, if you modify this document, you must indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use