HA NFS Cluster using Pacemaker and DRBD on RHEL/CentOS 8

Matt Kereczman;David Thomas;Michael Troutman

Version 1.2, 2022-10-25

Table of Contents

Chapter 1. Abstract

This guide can be used to deploy a high-availability (HA) two node NFS cluster on a LAN. To achieve this, we will use DRBD, Pacemaker, and Corosync on a RHEL/AlmaLinux/CentOS 8 server. With this solution set up, data transfer between a client system and an NFS share should not be interrupted by failure of a node. In this guide, we will set up the HA NFS cluster, simulate a node failure, and verify that our data transfer continues.

For the testing environment in this guide, we will use AlmaLinux 8 and the cluster stack software as packaged by LINBIT. Please contact the LINBIT sales team (sales@linbit.com) for access to this software.

Chapter 2. Assumptions

This guide assumes the following:

2.1. System Configuration

Hostname	LVM Device	Volume Group	DRBD Device	External Interface	External IP	Crossover Interface	Crossover IP
node-a	/dev/sdb	vg_drbd	lv_r0	eth0	192.168.10.201	eth1	172.16.0.201
node-b	/dev/sdb	vg_drbd	lv_r0	eth0	192.168.10.202	eth1	172.16.0.202

We'll need a virtual IP for services to run on. For this guide we will use 192.168.10.200

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.

Component	Protocol	Port
DRBD	TCP	7788
Corosync	UDP	5404, 5405

2.3. SELinux

If you have SELinux enabled, and you're having issues, consult your distributions documentation for how to properly configure it, or disable it (not recommended).

Chapter 3. Installation and Configuration

It is necessary to register our two NFS cluster nodes with LINBIT and bring in DRBD and Pacemaker software from official LINBIT repositories. For our NFS cluster setup, we will need to enable these LINBIT repositories:

- pacemaker-2
- drbd-9

3.1. Registering Nodes and Configuring Package Repositories

You will install DRBD and other cluster stack software that you may need from LINBIT's repositories. To access those repositories you will need to have been set up in LINBIT's system and have access to the LINBIT Customer Portal. If you have not been set up in LINBIT's system, please contact a sales team member: sales@linbit.com.

Once you have access to the Customer Portal, you can register your cluster nodes and configure repository access by using LINBIT's Python command line script. See the "REGISTER NODES" section of the Customer Portal for details about this script.

To download and run the LINBIT configuration script, enter the following commands on all nodes, one node at a time:

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

Script must be run as superuser.

If the error message `no python interpreter found :-(` is displayed when running `linbit-manage-node.py`, enter the command `dnf install python3` to install Python 3.

The script will prompt you to enter your LINBIT Customer Portal username and password. After validating your credentials, the script will list clusters and nodes (if you have any already registered) that are associated with your account.

Enter which cluster you want to register the current node with. You will then be asked a series of questions regarding which repositories you want to enable.

```
    1) pacemaker-2(Disabled)
    2) drbd-proxy-3.2(Disabled)
    3) drbd-9.0-only(Disabled)
    4) drbd-9.0(Disabled)
    5) drbd-9(Disabled)
```

The "drbd-9" repository includes the latest version, presently 9.1.x. As DRBD 9 is the latest version, it contains newer packages than the other two DRBD repositories. The "drbd-9.0" repository holds 9.0.x packages, for those who want to remain on that branch. The "drbd-9.0-only" contains the DRBD packages only (no LINSTOR).

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

After the script completes, you should be able to enter `dnf info kmod-drbd` and see the DNF package manager pulling package information from LINBIT repositories.

Before installing packages, be sure to only pull the cluster stack packages from LINBIT repositories.

3.1.1. Excluding Packages from Mainstream Repositories

To ensure that you only pull cluster packages from LINBIT repositories, add the following exclude line to the system package repository files:

```
exclude=cluster* corosync* drbd kmod-drbd libqb* pacemaker* resource-agents*
```

The `x86_64` architecture repositories are used in the following examples. Adjust appropriately if your system architecture is different.

3.1.2. Excluding Packages from RHEL 8 Repositories

The default location for all repositories in RHEL 8 is `/etc/yum.repos.d/redhat.repo`. Add the exclude line to both the `[rhel-8-for-x86_64-baseos-rpms]` and `[rhel-8-for-x86_64-appstream-rpms]` repository sections within the `.repo` file. The modified repository configuration should look like this:

```
# cat /etc/yum.repos.d/redhat.repo

[rhel-8-for-x86_64-baseos-rpms]
name = Red Hat Enterprise Linux 8 for x86_64 - BaseOS (RPMs)
...
exclude=cluster* corosync* drbd kmod-drbd libqb* pacemaker* resource-agents*

[rhel-8-for-x86_64-appstream-rpms]
name = Red Hat Enterprise Linux 8 for x86_64 - AppStream (RPMs)
...
exclude=cluster* corosync* drbd kmod-drbd libqb* pacemaker* resource-agents*
```

If the *Red Hat High Availability Add-On* is enabled, either add the exclude line to the `[rhel-8-for-x86_64-highavailability-rpms]` section or consider disabling the repository. LINBIT provides most of the packages available in the HA repository.

3.1.3. Excluding Packages from CentOS 8 Repositories

Add the exclude line to both the `[BaseOS]` section of `/etc/yum.repos.d/CentOS-Base.repo` as well as the `[AppStream]` section of `/etc/yum.repos.d/CentOS-AppStream.repo` repository files. The modified repository configuration should look like this:

```
# cat /etc/yum.repos.d/CentOS-Base.repo

[BaseOS]
name=CentOS-$releasever - Base
...
exclude=cluster* corosync* drbd kmod-drbd libqb* pacemaker* resource-agents*
```

```
# cat /etc/yum.repos.d/CentOS-AppStream.repo

[AppStream]
name=CentOS-$releasever - AppStream
...
exclude=cluster* corosync* drbd kmod-drbd libqb* pacemaker* resource-agents*
```

If the `[HighAvailability]` repo is enabled in `/etc/yum.repos.d/CentOS-HA.repo`, add the exclude line to the `[HighAvailability]` section or else consider disabling the repository. LINBIT provides most of the packages available in the HA repository.

3.1.4. Excluding Packages from AlmaLinux 8 Repositories

The default location for all repositories in AlmaLinux 8 is `/etc/yum.repos.d/almalinux.repo`. Add the exclude line to both the `[baseos]` and `[appstream]` sections of this file.

```
# cat /etc/yum.repos.d/almalinux.repo

[baseos]
name=AlmaLinux $releasever - BaseOS
...
exclude=cluster* corosync* drbd kmod-drbd libqb* pacemaker* resource-agents*

[appstream]
name=AlmaLinux $releasever - AppStream
...
exclude=cluster* corosync* drbd kmod-drbd libqb* pacemaker* resource-agents*
...
```

If the `[HighAvailability]` repo is enabled, `enabled=1`, in `/etc/yum.repos.d/almalinux-ha.repo`, add the exclude line to the `[ha]` section or else consider disabling the repository. LINBIT provides most of the packages available in the HA repository.

3.2. Install DRBD

Install DRBD, the DRBD kernel module, and the Cluster Management Shell, using the following command:

```
# dnf install drbd kmod-drbd crmsh
```

As Pacemaker will be responsible for starting the DRBD service, prevent DRBD from starting at boot:

```
# systemctl disable drbd
```

3.3. Configure DRBD

Now that we’ve installed DRBD, we’ll need to create our resource configuration file. To do this, create `/etc/drbd.d/r0.res` with the following contents:

```
resource r0 {
        protocol C;
        device    /dev/drbd0;
        disk      /dev/vg_drbd/lv_r0;
        meta-disk internal;
        on node-a {
            address 172.16.0.201:7788;
        }
        on node-b {
            address 172.16.0.202:7788;
        }
}
```

This is a bare-bones configuration file. There are many settings you can add and adjust to increase performance. See the *DRBD User's Guide* for more information.

Create the resource metadata by issuing the following command:

```
# drbdadm create-md r0
initializing activity log
initializing bitmap (32 KB) to all zero
Writing meta data...
New drbd meta data block successfully created.
success
```

This command should complete without any warnings - if you get messages about data being detected, and choose to proceed, you will lose data.

If you get any error messages when running the above `drbdadm` command, you can verify your DRBD configuration by entering the following command: `drbdadm dump all`.

Bring the device up on both nodes and verify their states by entering the following commands:

```
# drbdadm up r0
# drbdadm status
r0 role:Secondary
  disk:Inconsistent
  node-b role:Secondary
    peer-disk:Inconsistent
```

You can see in the above output that the resource is connected, but in an inconsistent state. To have your data replicated, you'll need to put the resource into a consistent state. There are two options:

1. Do a full sync of the device, which could potentially take a long time depending upon the size of the disk.
2. Skip the initial sync.

As we know this is a new setup with "just created" metadata and without any existing data on our device, we'll use option 2 and skip the initial sync. Enter the following commands on **node-a**:

```
# drbdadm new-current-uuid --clear-bitmap r0/0
# drbdadm status
r0 role:Secondary
  disk:UpToDate
  node-b role:Secondary
    peer-disk:UpToDate
```

3.4. Create a Filesystem on Our DRBD Backed Device

After DRBD is initialized, we'll need to make 'node-a' Primary and create a filesystem on the DRBD device that we configured earlier as resource `r0`.

First, make the `node-a` Primary for resource `r0` by entering the following command on `node-a`:

```
# drbdadm primary r0
```

Next, on `node-a`, create a file system on the backing device.

```
# mkfs.xfs /dev/drbd0
```

You only need to create the file system on 'node-a', and **not** on 'node-b'. DRBD will take care of the replication.

Create the following directory on **both** 'node-a' and 'node-b':

```
# mkdir /mnt/drbd
```

This concludes the initial setup of the DRBD device. We will next configure Corosync on each of our nodes.

3.5. Install Pacemaker and Corosync

This section will cover installing Pacemaker and Corosync. We will use Pacemaker as our cluster resource manager (CRM). Corosync acts a messaging layer, providing information to Pacemaker about the state of our cluster nodes.

Enter the following commands to install the necessary packages and then enable the Pacemaker and Corosync services to start when the system boots:

```
# dnf install pacemaker corosync

# systemctl enable pacemaker
Created symlink /etc/systemd/system/multi-user.target.wants/pacemaker.service to
/usr/lib/systemd/system/pacemaker.service.

# systemctl enable corosync
Created symlink /etc/systemd/system/multi-user.target.wants/corosync.service to
/usr/lib/systemd/system/corosync.service.
```

3.5.1. Configure Corosync

Create and edit the file `/etc/corosync/corosync.conf`. It should look like this:

```
totem {
  version: 2
  secauth: off
  cluster_name: cluster
  transport: knet
  rrp_mode: passive
}

nodelist {
  node {
    ring0_addr: 172.16.0.201
    ring1_addr: 192.168.10.201
    nodeid: 1
    name: node-a
  }
  node {
    ring0_addr: 172.16.0.202
    ring1_addr: 192.168.10.202
    nodeid: 2
    name: node-b
  }
}

quorum {
  provider: corosync_votequorum
  two_node: 1
}

logging {
  to_syslog: yes
}
```

Now that Corosync has been configured, start the Corosync and Pacemaker services:

```
# systemctl start corosync
# systemctl start pacemaker
```

Repeat the preceding Pacemaker and Corosync installation and configuration steps on each cluster node. Verify that everything has been started and is working correctly by entering the following `crm_mon` command. You should get output similar to this:

```
# crm_mon -rf -n1
Cluster Summary:
    * Stack: corosync
    * Current DC: node-a (version 2.0.5.linbit-1.0.el8-ba59be712) - partition with quorum
    * Last updated: Mon Feb 14 00:03:44 2022
    * Last change: Sun Feb 13 04:18:45 2022 by root crmd on node-a
    * 2 nodes configured

Node List:
    * Online: [ node-a node-b ]

Inactive resources:

Migration Summary:
    * Node node-a:
    * Node node-b:
```

3.6. Additional Cluster-Level Configuration

Since we only have two nodes, we will need to tell Pacemaker to ignore quorum. Run the following commands from either cluster node (but not both):

```
# crm configure property no-quorum-policy=ignore
```

Furthermore, we will not be configuring node level fencing (aka STONITH) in this guide. Disable STONITH using the following command:

```
# crm configure property stonith-enabled=false
```

Fencing/STONITH is an important part of HA clustering and should be used whenever possible. Disabling STONITH will lend the cluster vulnerable to split-brains and potential data corruption or loss.

For more information on Fencing and STONITH, you can review the ClusterLabs page on STONITH or contact the experts at LINBIT.

Chapter 4. Configure Pacemaker for HA NFS

Now that we have initialized our cluster, we can begin configuring Pacemaker to manage our resources. Instead of editing a configuration file, you can make these configuration changes using the `crm` shell. To start the `crm` shell, enter `crm configure`.

In this guide, we use the `crm configure` command, without any arguments, to work directly with our "live" cluster configuration. In a production environment, this may be risky. Instead, you may wish to work with `crm_shadow` configurations first. This allows you to test configurations in a sandbox environment, before changing your "live" system.

4.1. Configure DRBD Resources

DRBD is the first resource we will configure in Pacemaker. Use the following commands to pull a working version of the Cluster Information Base (CIB), configure the DRBD primitive (`p_drbd_r0`) and the Master/Slave set for the DRBD resource, and finally verify and commit the configuration changes:

```
# crm configure
crm(live)configure# primitive p_drbd_r0 ocf:linbit:drbd \
params drbd_resource=r0 \
op start interval=0s timeout=240s \
op stop interval=0s timeout=100s \
op monitor interval=31s timeout=20s role=Slave \
op monitor interval=29s timeout=20s role=Master
crm(live)configure# ms ms_drbd_r0 p_drbd_r0 meta master-max=1 \
master-node-max=1 clone-max=2 clone-node-max=1 notify=true
crm(live)configure# verify
crm(live)configure# commit
crm(live)configure# quit
```

Now, if we run `crm_mon` to view the cluster state, we should see that Pacemaker is managing our DRBD device:

```
# crm_mon
Cluster Summary:
    * Stack: corosync Current DC: node-b (version 2.0.5.linbit-1.0.el8-ba59be712) - partition
with quorum
    * Last updated: Mon Feb 14 01:44:19 2022
    * Last change: Sun Feb 13 04:18:45 2022 by root via cibadmin on node-a
    * 2 nodes configured
    * 2 resources configured

Online: [ node-a node-b ]

Active resources:

    * Clone Set: ms_drbd_r0 [p_drbd_r0] (promotable)
        * Masters: [ node-b ]
        * Slaves: [ node-a ]
```

4.2. Configure the Filesystem Primitive

With DRBD running, we can now configure our filesystem within Pacemaker. We will need to configure `colocation` and `order` constraints to ensure that the filesystem is mounted where DRBD is Primary, and only after DRBD has been promoted to Primary:

```
# crm configure
crm(live)configure# primitive p_fs_drbd0 ocf:heartbeat:Filesystem \
params device=/dev/drbd0 directory=/mnt/drbd fstype=xfs \
options=noatime,nodiratime \
op start interval="0" timeout="60s" \
op stop interval="0" timeout="60s" \
op monitor interval="20" timeout="40s"
crm(live)configure# order o_drbd_r0-before-fs_drbd0 \
ms_drbd_r0:promote p_fs_drbd0:start
crm(live)configure# colocation c_fs_drbd0-with_drbd-r0 \
inf: p_fs_drbd0 ms_drbd_r0:Master
crm(live)configure# verify
crm(live)configure# commit
crm(live)configure# quit
# crm_mon
```

Our `crm_mon` output should now look similar to this:

```
Cluster Summary:
...
    * 2 nodes configured
    * 3 resource instances configured

Online: [ node-a node-b ]

Active resources:
    * Clone Set: ms_drbd_r0 [p_drbd_r0] (promotable)
    * Masters: [ node-b ]
    * Slaves: [ node-a ]
    * p_fs_drbd0 (ocf::heartbeat:Filesystem):    Started node-b
```

If you type 'mount' on node-b, you should see the DRBD device mounted at /mnt/drbd.

4.3. Configure the NFS Service and Exports

We will now configure our NFS server and the exported filesystem.

Make sure that the 'nfs-utils' and 'rpcbind' packages are installed on both nodes, and that 'rpcbind' is enabled to start at boot:

```
# dnf install nfs-utils rpcbind
# systemctl enable rpcbind
# systemctl start rpcbind
```

Since the filesystem needs to be mounted locally before the NFS server can export it to clients on the network, we will need to set the appropriate ordering and colocation constraints for our resources: NFS will start on the node where the filesystem is mounted, and only after it's been mounted; then the 'exportfs' resources can start on the node where the filesystem is mounted.

In the example below, we have given meaningful names to both our `order` and `colocation` constraints, `o_fs_drbd0-before-nfsserver` and `c_nfsserver-with-fs_drbd0`.

First, we will need to define the primitive for the NFS server. The NFS server requires a directory to store its special files. This needs to be placed on our DRBD device, since it needs to be present where and when the NFS server starts, to allow for smooth failover.

Run the following commands on one of the nodes:

```
# crm configure
crm(live)configure# primitive p_nfsserver ocf:heartbeat:nfsserver \
params nfs_shared_infodir=/mnt/drbd/nfs_shared_infodir nfs_ip=192.168.10.200 \
op start interval=0s timeout=40s \
op stop interval=0s timeout=20s \
op monitor interval=10s timeout=20s
crm(live)configure# order o_fs_drbd0-before-nfsserver inf: p_fs_drbd0 p_nfsserver
crm(live)configure# colocation c_nfsserver-with-fs_drbd0 inf: p_nfsserver p_fs_drbd0
crm(live)configure# verify
crm(live)configure# commit
crm(live)configure# quit
# crm_mon
```

After a few seconds, you should see the NFS server resource start on 'node-b':

```
# crm_mon
Cluster Summary:
...
    * 2 nodes configured
    * 4 resources configured

Node List:
    * Online: [ node-a node-b ]

Active Resources:
    * Master/Slave Set: ms_drbd_r0 [p_drbd_r0]
        * Masters: [ node-b ]
        * Slaves: [ node-a ]
        * p_fs_drbd0 (ocf::heartbeat:Filesystem):       Started node-b
        * p_nfsserver       (ocf::heartbeat:nfsserver): Started node-b
```

With the NFS server running, we can create and configure our exports (from whichever node has 'p_nfsserver' started in your cluster):

```
# mkdir -p /mnt/drbd/exports/dir1
# chown nobody:nobody /mnt/drbd/exports/dir1
# crm configure
crm(live)configure# primitive p_exportfs_dir1 ocf:heartbeat:exportfs \
params clientspec=192.168.10.0/24 directory=/mnt/drbd/exports/dir1 fsid=1 \
unlock_on_stop=1 options=rw,sync \
op start interval=0s timeout=40s \
op stop interval=0s timeout=120s \
op monitor interval=10s timeout=20s
crm(live)configure# order o_nfsserver-before-exportfs-dir1 inf: p_nfsserver p_exportfs_dir1
crm(live)configure# colocation c_exportfs-with-nfsserver inf: p_exportfs_dir1 p_nfsserver
crm(live)configure# verify
crm(live)configure# commit
crm(live)configure# quit
```

You should now be able to use the `showmount` command from a client system to see the exported directories on the current Primary node (`node-a` in our example):

```
# showmount -e node-a
Export list for node-a:
/mnt/drbd/exports/dir1 192.168.10.0/24
```

4.4. Configure the Virtual IP

The Virtual IP (VIP) will provide consistent client access to the NFS export if one of our cluster nodes should fail. To do this, we will need to define a `colocation` constraint so that the VIP resource will always start on the node where the NFS export is currently active:

```
# crm configure
crm(live)configure# primitive p_virtip_dir1 ocf:heartbeat:IPaddr2 \
params ip=192.168.10.200 cidr_netmask=24 \
op monitor interval=20s timeout=20s \
op start interval=0s timeout=20s \
op stop interval=0s timeout=20s
crm(live)configure# order o_exportfs_dir1-before-p_virtip_dir1 \
inf: p_exportfs_dir1 p_virtip_dir1
crm(live)configure# colocation c_virtip_dir1-with-exportfs-dir1 \
inf: p_virtip_dir1 p_exportfs_dir1
crm(live)configure# verify
crm(live)configure# commit
crm(live)configure# quit
```

Now we should be able to use the `showmount` command from a client system, specifying the Virtual IP, and see the same output as we saw above:

```
# showmount -e 192.168.10.200
Export list for 192.168.10.200:
/mnt/drbd/exports/dir1 192.168.10.0/24
```

Chapter 5. Test Cluster Failover

There are many ways we could test the persistence of our cluster's NFS export in a failover scenario. We could failover while copying a file from a client system to our NFS export. Or we could have a client system play an audio file stored on the NFS export during a failover, etc.

In this guide, we'll describe how to simulate a file copy during a failover scenario. First, we'll create a "large" file in the mounted export directory using the dd utility, while failing over.

Mount the exported filesystem on a **client system**, and use dd to create a 1GiB file named 'write_test.out':

```
# mkdir /mnt/test_nfs_mount
# mount 192.168.10.200:/mnt/drbd/exports/dir1 /mnt/test_nfs_mount
# dd if=/dev/zero of=/mnt/test_nfs_mount/write_test.out bs=1M count=1024
```

Before the dd command on the client completes, hard reset the Primary node with the following command:

```
# echo b > /proc/sysrq-trigger
```

The node should reboot immediately, causing the cluster to migrate all services to the peer node. The failover should not interrupt the dd command on the client system and the command should complete the file write without error.

To further verify our failover scenario, we can use the `drbdadm status` command on our Primary node, i.e., the node that we just hard reset:

```
# drbdadm status
r0 role:Secondary
  disk:UpToDate
    node-b role:Primary
        peer-disk:UpToDate
```

This output confirms that during the failover, our DRBD, Corosync, and Pacemaker HA solution promoted node-b to a Primary role when we hard reset node-a.

Congratulations! You've set up a high-availability NFS cluster using DRBD, Corosync, and Pacemaker! We'll leave configuring additional exports as an exercise for the reader. You may also want to try other scenarios where a client system is accessing an NFS cluster export during a failover.

Chapter 6. Conclusion

This guide was created with RHEL/CentOS/AlmaLinux systems in mind. If you have any questions regarding setting up a high-availability NFS cluster in your environment, you can contact the experts at LINBIT.

Appendix A: Additional Information and Resources

- LINBIT's GitHub Organization: https://github.com/LINBIT/
- Join LINBIT's Community on Slack: https://www.linbit.com/join-the-linbit-drbd-linstor-slack/
- The DRBD® and LINSTOR® User's Guide: https://docs.linbit.com/
- The DRBD® and LINSTOR® Mailing Lists: https://lists.linbit.com/
    - drbd-announce: Announcements of new releases and critical bugs found
    - drbd-user: General discussion and community support
    - drbd-dev: Coordination of development

Appendix B: Legalese

B.1. Trademark Notice

B.2. License Information