iSCSI High Availability Clustering Using DRBD and Pacemaker on RHEL 9

Matt Kereczman, Brian Hellman, Hayley Swimelar, David Thomas

Version 1.1, 2024-03-21

Table of Contents

Chapter 1. Introduction

This guide outlines the configuration of a highly available iSCSI storage cluster using DRBD®, Pacemaker, and Corosync. iSCSI provides access to block devices via TCP/IP, which allows storage to be accessed remotely using standard networks.

An iSCSI initiator (client) connects to an iSCSI target (server) and accesses a Logical Unit Number (LUN). Once a LUN is connected, it will function as a normal block device to the client.

This guide was written for RHEL 9, using the cluster stack software as packaged by LINBIT®.

Chapter 2. Assumptions

This guide assumes the following:

2.1. System Configurations

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

You will need a virtual IP for services to run on. For this guide you will use 192.168.10.200. Instructions for setting this up will be shown later in this guide.

2.2. Firewall Configuration

Refer to your firewall documentation for how to open or 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 are having issues, consult your distribution's documentation for how to properly configure it, or disable it (not recommended).

Chapter 3. Installation and Configuration

3.1. Registering Nodes with LINBIT and Configuring Package Repositories

You will install DRBD and dependencies that you may need from LINBIT's customer 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, or if you want an evaluation account, you can contact a sales team member: sales@linbit.com.

3.1.1. Using the LINBIT Customer Portal to Register Nodes

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

Downloading and Running the LINBIT Manage Nodes Helper Script

To download and run the LINBIT helper script to register your nodes and configure LINBIT repository access, 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
```

The 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 -y 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.

Joining Nodes to a Cluster

Select the cluster that you want to register the current node with. If you want the node to be the first node in a new cluster, select the "new cluster" option.

Saving the Registration and Repository Configurations to Files

To save the registration information on your node, confirm the writing of registration data to a JSON file, when the helper script prompts you to.

```
Writing registration data:
--> Write to file (/var/lib/drbd-support/registration.json)? [y/N]
```

To save the LINBIT repository configuration to a file on your node, confirm the writing of a `linbit.repo` file, when the helper script prompts you to.

```
--> Write to file (/etc/yum.repos.d/linbit.repo)? [y/N]
```

3.2. Enabling Access to LINBIT Repositories

After registering a node by using the LINBIT manage nodes helper script and joining the node to a cluster, the script will show you a menu of LINBIT repositories. For RHEL 9, the possible repositories are:

```
1) pacemaker-2(Disabled)
2) drbd-proxy-3.2(Disabled)
3) drbd-9(Disabled)
```

The `drbd-9` repository includes the latest DRBD 9 version. It also includes other LINBIT software packages, including LINSTOR®, DRBD Reactor, LINSTOR GUI, OCF resource agents, and others.

Following the helper script's instructions allows you to enable or disable the repositories that you need or do not need for your use case. For the purposes of this guide, you will need to enable the following repositories:

- `pacemaker-2`
- `drbd-9`

You can use the high-availability (HA) repository in either RHEL or AlmaLinux, rather than LINBIT's `pacemaker-2` repository. If you choose to use the RHEL or AlmaLinux HA repository, you will need to adjust the instructions in the upcoming Excluding Packages section so that you do not exclude packages from the HA repository.

3.3. Installing LINBIT's Public Key and Verifying LINBIT Repositories

After enabling and disabling LINBIT repositories and confirming your selection, be sure to respond **yes** to the questions about installing LINBIT's public key to your keyring and writing the repository configuration file.

3.3.1. Verifying LINBIT Repositories

After the LINBIT manage nodes helper script completes, you can verify that you enabled LINBIT repositories by using the `dnf info` command. For example, to verify that you enabled the `drbd-9` repository, enter the command:

```
# dnf info kmod-drbd
```

Output from the command should show the DNF package manager pulling package information from LINBIT repositories.

To use this method to verify the LINBIT Pacemaker repository, you will first need to exclude some packages within RHEL repositories, otherwise the `dnf info` command might show RHEL repositories as sources for the Pacemaker or related packages.

At this point though, you can enter the command `dnf repolist` to show that at least a configuration exists on the system for the repositories that output from the `dnf repolist` command shows.

3.4. Excluding Packages from Red Hat or AlmaLinux Repositories

Before installing packages, be sure to only pull the cluster stack packages from LINBIT repositories. To do this, you will need to exclude certain packages, in the RHEL or AlmaLinux repositories, that overlap with packages in the LINBIT customer repositories.

The commands that follow insert an "exclude" line after the occurrence of every enabled repository line in the given files.

3.4.1. Excluding Packages from Red Hat Repositories

To exclude the necessary packages from enabled RHEL repositories, enter the commands:

```
# RH_REPOS="`ls /etc/yum.repos.d/*.repo|grep -v linbit`"
# PKGS="cluster\* corosync\* drbd kmod-drbd libqb\* pacemaker\* pcs resource-agents\*"
# for file in $RH_REPOS; do sed -i "/^enabled[ =]*1/a exclude=$PKGS" $file; done
```

3.4.2. Excluding Packages from AlmaLinux Repositories

To exclude the necessary packages from enabled AlmaLinux repositories, enter the commands:

```
# AL_REPOS="`ls /etc/yum.repos.d/*.repo|grep -v linbit`"
# PKGS="cluster\* corosync\* drbd kmod-drbd libqb\* pacemaker\* pcs resource-agents\*"
# for file in $AL_REPOS; do sed -i "/^enabled[ =]*1/a exclude=$PKGS" $file; done
```

If you are using the RHEL or AlmaLinux "High Availability" repository, you could choose to install Pacemaker and related packages from that repository, rather than the LINBIT `pacemaker-2` repository. In that case, you would have to adjust the exclude statement editing instructions here accordingly.

3.5. Install DRBD

Install DRBD and the DRBD kernel module by entering the following command:

```
# dnf install -y drbd kmod-drbd
```

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

```
# systemctl disable drbd
```

3.6. Configure DRBD

Now that you have installed DRBD, you will need to create your 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 minimally configured resource configuration file. There are many settings you can add and

adjust to increase performance. See the *DRBD User 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
```

As the preceding output shows, the resource is connected, but in an inconsistent state. To have your data replicated, you will 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.

Because you know this is a new setup with "just created" metadata and without any existing data on your device, you can use the second option 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
```

After installing DRBD and preparing your DRBD resource on both nodes, you are ready to install and configure cluster resource manager (CRM) software components that will make your DRBD resource highly available.

Starting in RHEL 9, LVM device filtering is controlled by automatically generated `system.devices` files stored in `/etc/lvm/devices/`. These files are generated when creating physical volumes, and are intended to more accurately identify and track physical volumes associated with volume groups on a system. Device filtering is an important concept when exporting logical volumes which could be used as a physical volumes on remote systems. For more information on managing `system.devices`, see `man lvmdevices`.

3.7. Installing Pacemaker, Corosync, and the Pacemaker/Corosync Configuration System

This section will cover installing Pacemaker and Corosync. You will use Pacemaker as your cluster resource manager (CRM). Corosync acts as a messaging layer, providing information to Pacemaker about the state of your cluster nodes. The `pcs` command-line interface, and the `pcs` daemon (`pcsd`) are used to create and manage the cluster configurations.

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

```
# dnf install -y pacemaker corosync pcs

# 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.

# systemctl enable pcsd --now
Created symlink /etc/systemd/system/multi-user.target.wants/pcsd.service →
/usr/lib/systemd/system/pcsd.service.
```

3.7.1. Creating Initial Cluster Configurations

Begin authenticating the cluster nodes by configuring the same password for the `hacluster` user on each host:

```
# echo 'secretpassword' | passwd --stdin hacluster
Changing password for user hacluster.
passwd: all authentication tokens updated successfully.
```

Run the following command from one node in the cluster. Make sure `pcsd` is running on all hosts in the cluster before running the following command:

```
# pcs host auth -u hacluster -p secretpassword node-a node-b
node-b: Authorized
node-a: Authorized
```

Create and distribute the cluster configuration to each host using `pcs` from the same host in the cluster as the previous command:

```
# pcs cluster setup cluster \
    node-a addr=172.16.0.201 addr=192.168.10.201 \
    node-b addr=172.16.0.202 addr=192.168.10.202
```

Output from the previous command will show something similar to the following:

```
Destroying cluster on hosts: 'node-a', 'node-b'...
node-a: Successfully destroyed cluster
node-b: Successfully destroyed cluster
Requesting remove 'pcsd settings' from 'node-a', 'node-b'
node-a: successful removal of the file 'pcsd settings'
```

```
node-b: successful removal of the file 'pcsd settings'
Sending 'corosync authkey', 'pacemaker authkey' to 'node-a', 'node-b'
node-a: successful distribution of the file 'corosync authkey'
node-a: successful distribution of the file 'pacemaker authkey'
node-b: successful distribution of the file 'corosync authkey'
node-b: successful distribution of the file 'pacemaker authkey'
Sending 'corosync.conf' to 'node-a', 'node-b'
node-b: successful distribution of the file 'corosync.conf'
node-a: successful distribution of the file 'corosync.conf'
Cluster has been successfully set up.
```

Now that Corosync has been configured, start the Corosync and Pacemaker services by using the following `pcs` command from one node in the cluster:

```
# pcs cluster start --all
node-a: Starting Cluster...
node-b: Starting Cluster...
```

Verify that everything has been started and is working correctly by entering the following `pcs status` command. You should get output similar to this:

```
# pcs status
Cluster name: cluster

WARNINGS:
No stonith devices and stonith-enabled is not false

Cluster Summary:
  * Stack: corosync
  * Current DC: node-a (version 2.1.2.linbit-4.el9-ada5c3b36e2) - partition with quorum
  * Last updated: Mon Feb 19 03:27:22 2024
  * Last change:  Mon Feb 19 03:22:13 2024 by hacluster via crmd on node-a
  * 2 nodes configured
  * 0 resource instances configured

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

Full List of Resources:
  * No resources

Daemon Status:
  corosync: active/enabled
  pacemaker: active/enabled
  pcsd: inactive/disabled
```

3.8. Additional Cluster-Level Configuration

In a highly available two-node cluster using DRBD, you should make some recommended Pacemaker configurations:

- Disable STONITH.
- Set Pacemaker's "no quorum policy" to ignore the loss of quorum.
- Set the default resource stickiness to 200.

To make these configurations, enter the following `pcs` commands from any one (but only one) node in the cluster:

```
# pcs property set stonith-enabled=false
```

```
# pcs property set no-quorum-policy=ignore
# pcs resource defaults update resource-stickiness=200
```

To verify your configuration changes, enter the following commands:

```
# pcs property config stonith-enabled
# pcs property config no-quorum-policy
# pcs resource defaults
```

Output from the commands should show:

```
Cluster Properties:
 stonith-enabled: false
Cluster Properties:
 no-quorum-policy: ignore
Meta Attrs: build-resource-defaults
  resource-stickiness=200
```

You could also parse the output of the `pcs config show` command to verify the property and resource default values that you set.

STONITH is not strictly necessary because DRBD is a shared-nothing solution. However, it is highly recommended to prevent split-brains, and potential loss of data on the split-brain victim. For brevity, this guide disables STONITH. An excellent guide on STONITH and its configuration can be found on ClusterLab's site, or you can contact the experts at LINBIT for more information.

3.9. Selecting and Installing the iSCSI Stack and Dependencies

There is more than one iSCSI implementation you can choose from when building your highly available iSCSI cluster. This guide will describe two iSCSI implementations: Linux-IO (LIO) and the Generic SCSI Target Subsystem for Linux (SCST). Both LIO and SCST operate within kernel space. They also both support iSCSI persistent reservations, which is an important feature when attaching highly available initiators such as hypervisor clusters.

LIO is the iSCSI implementation that the Linux kernel community chose as the SCSI target subsystem that is included in the mainline Linux kernel.

SCST is a mature iSCSI implementation that is used in many iSCSI appliances, including LINBIT's VSAN since version 0.17.0.

3.9.1. Option A: LIO

TargetCLI is an interactive shell used to manage the Linux-IO (LIO) target. LIO is the SCSI target that has been included with the Linux kernel since 2.6.38, which makes the installation of its utilities and dependencies relatively simple.

TargetCLI can be installed using the following command on both nodes:

```
# dnf install targetcli
```

3.9.2. Option B: SCST

The SCST project consists of a kernel space core, device handlers, target drivers, and the `scstadmin` user space utility for managing its core components. All of which can be built from source, by following the instructions found on the project's GitHub repository.

By following the installation instructions below, you will install all the necessary components for building and using SCST with Pacemaker.

Enter and run all the commands in this section on both nodes.

ELRepo, the RPM repository for Enterprise Linux packages that are not included in the standard RHEL distribution's repositories, is needed for installing DKMS. Development tools and other dependencies for building SCST's RPMs must also be installed:

```
# dnf install -y https://dl.fedoraproject.org/pub/epel/epel-release-latest-9.noarch.rpm
# dnf groupinstall -y "Development Tools"
# dnf install -y kernel-devel perl perl-Data-Dumper perl-ExtUtils-MakeMaker rpm-build dkms git
```

After installing build dependencies, you can build and install the SCST packages:

```
# git clone https://github.com/SCST-project/scst
# cd scst/
# make rpm-dkms
# cd ~/
# dnf install -y /usr/src/packages/RPMS/x86_64/scst*
```

Finally, enter the commands below to create the necessary configuration for loading the SCST kernel module, create a systemd unit file for an `iscs-scst` service, and reload systemd services:

```
# echo -n "" > /etc/modules-load.d/scst.conf
# for m in iscsi-scst scst scst_vdisk; do
    echo $m >> /etc/modules-load.d/scst.conf
    modprobe $m
  done

# cat << EOF > /etc/systemd/system/iscsi-scst.service
[Unit]
Description=iSCSI SCST Target Daemon
Documentation=man:iscsi-scstd(8)
After=network.target
Before=scst.service
Conflicts=shutdown.target

[Service]
EnvironmentFile=-/etc/sysconfig/scst
PIDFile=/var/run/iscsi-scstd.pid
ExecStartPre=/sbin/modprobe iscsi-scst
ExecStart=/sbin/iscsi-scstd $ISCSID_OPTIONS

[Install]
WantedBy=multi-user.target
EOF

# systemctl daemon-reload
# systemctl enable --now iscsi-scst
```

3.10. Creating an Active/Passive iSCSI Configuration

An active/passive iSCSI Target consists of the following cluster resources:

- A DRBD resource to replicate data, which the cluster resource manager switches between Primary and Secondary roles on nodes in your cluster as is needed to maintain resource availability.

- A virtual, floating cluster IP address, allowing initiators to connect to the target no matter which physical node the target is running on.
- The iSCSI target itself.
- Two port blocking rules to prevent connection errors while the target is being brought up or down: one to block the iSCSI target and another to unblock it.
- One or more iSCSI Logical Units (LUNs), each corresponding to a Logical Volume in the LVM Volume Group.

The following Pacemaker configuration example assumes that `192.168.10.200` is the virtual IP address to use for a target with the iSCSI Qualified Name (IQN) `iqn.2024-01.com.example:drbd0`.

The target is to contain one Logical Unit: (LUN 0) mapping to `/dev/drbd0`.

To start configuring these resources, use `pcs` to save the current Pacemaker configuration to a file named `cib.txt` as the `root` user (or any non-`root` user that is part of the `haclient` group), by issuing the following command:

You only need to run the `pcs` commands shown below on one of your cluster nodes.

```
# pcs cluster cib cib.txt
```

Now create the DRBD resource primitive by running the following commands:

```
# pcs -f cib.txt resource create p_drbd_r0 ocf:linbit:drbd \
  drbd_resource="r0" \
  op monitor interval="29s" role="Promoted" \
  op monitor interval="31s" role="Unpromoted"
```

Running `pcs -f cib.txt cluster verify --full` after configuring each new Pacemaker resource is recommended: it will alert you if you have syntax errors in your configuration. You can edit the configuration using a text editor to correct errors, or use `pcs -f cib.txt resource delete <name>` to remove an erroneous resource from the configuration.

Next, create a multi-state clone resource corresponding to the DRBD resource `r0`:

```
# pcs -f cib.txt resource clone p_drbd_r0 ms_drbd_r0 \
  promoted-max=1 promoted-node-max=1 promotable=true \
  clone-max=2 clone-node-max=1 notify=true
```

You will now create a floating virtual IP address for the cluster to be managed by Pacemaker:

```
# pcs -f cib.txt resource create p_iscsi_ip0 ocf:heartbeat:IPaddr2 \
  ip="192.168.10.200" cidr_netmask="24" \
  op start timeout=20 \
  op stop timeout=20 \
  op monitor interval="10s"
```

iSCSI IQN names must be unique and should follow RFC 3720 standards.

```
# pcs -f cib.txt resource create p_iscsi_target_drbd0 ocf:heartbeat:iSCSITarget \
  iqn="iqn.2024-01.com.example:drbd0" portals="192.168.10.200:3260" \
  op start timeout=20 \
```

```
  op stop timeout=20 \
  op monitor interval=20 timeout=40
```

After establishing the iSCSI target and virtual IP address, next configure the Logical Units:

```
# pcs -f cib.txt resource create p_iscsi_lun_drbd0 ocf:heartbeat:iSCSILogicalUnit \
  target_iqn="iqn.2024-01.com.example:drbd0" lun=0 path="/dev/drbd0" scsi_sn="aaaaaaa0" \
  op start timeout=20 \
  op stop timeout=20 \
  op monitor interval=20 timeout=40
```

Both the `iSCSITarget` and `iSCSILogicalUnit` resource agents have an `implementation` parameter. Omitting the parameter causes the resource agents to determine which implementation to use based on the presence of implementation specific binary files. If you have multiple implementations installed, or simply want to be explicit in your resource definitions, you can set the `implementation` parameter equal to `scst` or `lio-t`.

Configure port blocking and unblocking. This will prevent an iSCSI initiator from receiving a "Connection refused" error during failover before the iSCSI target has successfully started.

```
# pcs -f cib.txt resource create p_iscsi_portblock_on_drbd0 ocf:heartbeat:portblock \
  ip=192.168.10.200 portno=3260 protocol=tcp action=block \
  op start timeout=20 \
  op stop timeout=20 \
  op monitor timeout=20 interval=20

# pcs -f cib.txt resource create p_iscsi_portblock_off_drbd0 ocf:heartbeat:portblock \
  ip=192.168.10.200 portno=3260 protocol=tcp action=unblock \
  op start timeout=20 \
  op stop timeout=20 \
  op monitor timeout=20 interval=20
```

To tie all of this together, create a resource group from the resources associated with your iSCSI Target:

```
# pcs -f cib.txt resource group add g_iscsi_drbd0 \
  p_iscsi_portblock_on_drbd0 p_iscsi_ip0 \
  p_iscsi_target_drbd0 p_iscsi_lun_drbd0 \
  p_iscsi_portblock_off_drbd0
```

This group, by default, is *ordered* and *colocated*, which means that the resources contained therein will always run on the same physical node, will be started in the order as specified, and stopped in reverse order.

Finally, you have to ensure that this resource group is started after the DRBD resource has started and set it to run on the same node where DRBD is in the Primary role:

```
# pcs -f cib.txt constraint order ms_drbd_r0 then g_iscsi_drbd0

# pcs -f cib.txt constraint colocation add g_iscsi_drbd0 with promoted ms_drbd_r0
```

Now, your configuration is complete, and you can apply it to the running cluster:

```
# pcs cluster cib-push cib.txt
```

3.11. Creating an Active/Active iSCSI Configuration

Configuring an active/active iSCSI target is similar to an active/passive target, with the exception of the following:

- There must be at least two DRBD resources to replicate data, which can run independently on separate nodes.
- One LVM Logical Volume to serve as the backing device for each DRBD resource.
- Each target must have its own floating cluster (virtual) IP address, allowing initiators to connect to the target no matter which physical node the target is running on.
- Each DRBD resource needs to have a resource group, plus order and colocation constraints similar to the single DRBD resource in the active/passive section.

Once these resource groups are configured, you can have them run on separate nodes to achieve an active/active cluster. This can be done by manually migrating the resource groups by using `pcs` commands, or by configuring the resources to prefer running on a particular node.

Chapter 4. Security Considerations and Data Integrity

There are a few more points worth mentioning, dealing with access control and data integrity.

4.1. Restricting Target Access by Initiator Address

Access to iSCSI targets should be restricted to specific initiators, identified by their iSCSI Qualified Name (IQN). Use the `allowed_initiators` parameter supported by the iSCSI Target Pacemaker resource agent.

Create a resource to include the `allowed_initiators` parameter, containing a space-separated list of initiator IQNs allowed to connect to this target. In the example below, access is granted to the initiator IQN `iqn.1994-05.com.example:5e6220f26ee`.

On a RHEL system, where the `iscsi-initiator-utils` package has been installed, you can find the initiator's IQN by entering:

```
cat /etc/iscsi/initiatorname.iscsi
```

To update parameters on the `p_iscsi_target_drbd0` resource use the following command:

```
# pcs resource update p_iscsi_target_drbd0 \
  allowed_initiators="iqn.1994-05.com.redhat:42615a3677 iqn.1994-05.com.redhat:27cac47b0ae"
```

Notice that the above command does not specify a configuration file using the `pcs -f <filename>` ... option. This means that `pcs` will make the changes to the resource configuration in the running cluster.

The target will immediately reconfigure and enable the access restrictions.

If initiators are connected to the target at the time of reconfiguration, and one of the connected initiators is not included in the `initiators` list for this resource, then those initiators will lose access to the target, possibly resulting in disruption on the initiator node. Use with care.

4.2. Dual-Primary Multipath iSCSI

Do not attempt to use Multipath iSCSI targets with dual-primary DRBD resources. Multipath iSCSI targets do not coordinate with each other and are not cluster aware.

Multipath iSCSI might appear to work with dual-primary DRBD resources under light testing, but will not be able to maintain consistent data under production loads or in the case of some failures, such as the loss of the replication link.

Chapter 5. Using Highly Available iSCSI Targets

This section describes some common usage scenarios for highly available iSCSI targets.

5.1. Connecting to iSCSI Targets From Linux

The recommended way of connecting to a highly available iSCSI Target from Linux is to use `iscsiadm` which you can install by entering the following command:

```
# dnf install -y iscsi-initiator-utils
```

After installing the `iscsi-initiator-utils` package, it is first necessary to start the iSCSI service, `iscsi`. To do so, issue the following command:

```
# systemctl start iscsi
```

Now you can start a discovery session on your target portal. Assuming your cluster IP address for the target is 192.168.10.200, you can do so by entering the following command:

```
# iscsiadm -m discovery -p 192.168.10.200 -t sendtargets
```

The output from this command should include the names of all targets you have configured.

```
192.168.10.200:3260,1 iqn.2017-10.com.example:drbd0
```

If a configured target does not appear in this list, check whether your initiator has been blocked from accessing this target by an initiator restriction (see Restricting Target Access by Initiator Address).

Finally, you can log in to the target, which will make all LUNs configured therein available as local SCSI devices:

```
# iscsiadm -m node -p 192.168.10.200 -T iqn.2024-01.com.example:drbd0 --login
```

Chapter 6. Feedback

Any questions or comments about this document are appreciated and encouraged.

For a public discussion about the concepts mentioned in this how-to guide, you are invited to subscribe and post to the drbd-user mailing list. See the drbd-user mailing list for details.

Appendix A: Additional Information and Resources

- LINBIT's GitHub Organization: https://github.com/LINBIT/
- Join the LINBIT Community Forums: https://forums.linbit.com
- The DRBD® and LINSTOR® User Guides: https://linbit.com/user-guides-and-product-documentation/
- 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

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

B.2. License Information