Highly Available GitLab Using DRBD and Pacemaker

Ryan Ronnander

1.0, 2020-12-01

Table of Contents

Chapter 1. Introduction

GitLab is a complete DevOps platform used by many organizations and developers. GitLab can be easily installed as a self-managed web application. This guide will demonstrate installing and configuring a highly available (HA) two node GitLab CE (Community Edition) environment using Pacemaker and DRBD.

Chapter 2. Installation Overview

Configure LINBIT repositories before installing DRBD, Pacemaker, and Corosync.

In order to create a highly available GitLab environment, you will need to install the following software packages.

- *GitLab* is a web-based DevOps platform written in Go, Ruby on Rails, and Vue.js. This guide will be using GitLab Community Edition.
- *Pacemaker* is a cluster resource management framework which you will use to automatically start, stop, monitor, and migrate resources. Distributions typically bundle Pacemaker in a package simply named `pacemaker`. This guide assumes that you are using Pacemaker 2.0.2 or greater installed from LINBIT's `[pacemaker-2]` repository.
- *Corosync* is the cluster messaging layer that Pacemaker uses for communication and membership. In distributions, the Corosync package is usually simply named `corosync`. This guide assumes that you are using Corosync version 3.0.2 or greater installed from LINBIT's `[pacemaker-2]` repository.
- *DRBD* is a kernel block-level synchronous replication facility which serves as an imported shared-nothing cluster building block. This guide assumes use of DRBD 9.0 installed from LINBIT's `[drbd-9.0]` repository. LINBIT support customers can get pre-compiled binaries from the official repositories. As always the source can be found at https://github.com/LINBIT/drbd.

You may be required to install packages other than the above-mentioned ones due to package dependencies. However, when using a package management utility such as `dnf`, these dependencies should be resolved automatically.

2.1. Register Nodes and Repository Configuration

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.

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 `# dnf info kmod-drbd` and see dnf pulling package information from LINBIT's repository.

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

To ensure we only pull cluster packages from LINBIT, we will need to add the following exclude line to our repository

files:

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

2.1.1. RHEL 8 Repository Configuration

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

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

```
# '/etc/yum.repos.d/redhat.repo' example:

[rhel-8-for-x86_64-baseos-rpms]
name = Red Hat Enterprise Linux 8 for x86_64 - BaseOS (RPMs)
baseurl = https://cdn.redhat.com/content/dist/rhel8/$releasever/x86_64/baseos/os
enabled = 1
gpgcheck = 1
gpgkey = file:///etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release
sslverify = 1
sslcacert = /etc/rhsm/ca/redhat-uep.pem
sslclientkey = /etc/pki/entitlement/<your_key_here>.pem
sslclientcert = /etc/pki/entitlement/<your_cert_here>.pem
metadata_expire = 86400
enabled_metadata = 1
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)
baseurl = https://cdn.redhat.com/content/dist/rhel8/$releasever/x86_64/appstream/os
enabled = 1
gpgcheck = 1
gpgkey = file:///etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release
sslverify = 1
sslcacert = /etc/rhsm/ca/redhat-uep.pem
sslclientkey = /etc/pki/entitlement/<your_key_here>.pem
sslclientcert = /etc/pki/entitlement/<your_cert_here>.pem
metadata_expire = 86400
enabled_metadata = 1
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.

2.1.2. CentOS 8 Repository Configuration

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:

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

[BaseOS]
name=CentOS-$releasever - Base
mirrorlist=http://mirrorlist.centos.org/?release=$releasever&arch=$basearch&repo=BaseOS&infra=$i
nfra
#baseurl=http://mirror.centos.org/$contentdir/$releasever/BaseOS/$basearch/os/
gpgcheck=1
enabled=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-centosofficial
exclude=cluster* corosync* drbd kmod-drbd libqb* pacemaker* resource-agents*
```

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

[AppStream]
name=CentOS-$releasever - AppStream
mirrorlist=http://mirrorlist.centos.org/?release=$releasever&arch=$basearch&repo=AppStream&infra
=$infra
#baseurl=http://mirror.centos.org/$contentdir/$releasever/AppStream/$basearch/os/
gpgcheck=1
enabled=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-centosofficial
exclude=cluster* corosync* drbd kmod-drbd libqb* pacemaker* resource-agents*
```

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

2.2. Installing DRBD

Install DRBD using the following command:

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

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

```
# systemctl disable drbd
```

2.3. Installing Pacemaker and Corosync

This section will cover installing Pacemaker and Corosync. Issue the following command to install and enable the necessary packages:

```
# dnf install pacemaker corosync crmsh

# 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.
```

2.4. Installing GitLab

Review GitLab's documentation for installing GitLab Community Edition. Some of the documented configuration steps are omitted from this guide.

Notification emails generated by GitLab often rely on `postfix` in GitLab installations. As this configuration step is optional and dependent on your environment, it will be omitted for the sake of simplicity.

In this section we will perform the GitLab installation and initial configuration steps on both nodes.

First, add the GitLab package repository:

```
# curl https://packages.gitlab.com/install/repositories/gitlab/gitlab-ce/script.rpm.sh | sudo
bash
```

Install GitLab Community Edition and any prerequisite packages:

```
# dnf install gitlab-ce
```

Next, initialize GitLab. This entire process is automated, it does not require user input and may take several minutes:

```
# gitlab-ctl reconfigure
```

At this point two GitLab instances are running and can be viewed within a web browser from either node's IP address, http://192.168.122.201/, and http://192.168.122.202/ as specified in the upcoming Initial Configuration chapter.

However, GitLab needs to be prevented from automatically starting at boot by `systemd`:

```
# systemctl disable gitlab-runsvdir.service
```

Any currently running Gitlab instance needs to stop in preparation for Pacemaker configuration:

```
# gitlab-ctl stop
# systemctl stop gitlab-runsvdir.service
```

Chapter 3. Initial Configuration

This section describes the initial configuration of a two node highly available GitLab environment in the context of the Pacemaker cluster manager.

3.1. System Configurations

Table 1. Node Configuration Overview

Hostname	LVM Device	Volume Group	DRBD Device	External Interface	External IP	Crossover Interface	Crossover IP
node-a	/dev/vdb	vg_drbd	lv_gitlab	enp1s0	192.168.122.201	enp9s0	172.16.0.201
node-b	/dev/vdb	vg_drbd	lv_gitlab	enp1s0	192.168.122.202	enp9s0	172.16.0.202

We'll need a virtual IP for GitLab web services to bind to. For this guide we will use `192.168.122.200`.

The Logical Volume Manager (LVM) commands used in the creation of this guide are included for your convenience, adjust accordingly:

```
# pvcreate /dev/vdb
  Physical volume "/dev/vdb" successfully created.

# vgcreate vg_drbd /dev/vdb
  Volume group "vg_drbd" successfully created

# lvcreate -L 20G -n lv_gitlab vg_drbd
  Logical volume "lv_gitlab" created.
```

3.2. Firewall Configuration

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

Table 2. Required Ports

Component	Protocol	Port
DRBD	TCP	7788
Corosync	UDP	5404, 5405
HTTP	TCP	80
HTTPS	TCP	443

3.3. 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).

3.4. Configuring DRBD

First, it is necessary to configure a DRBD resource to serve as a backing device for GitLab's underlying data. The suggested method is to use individual logical volumes as the backing device for each DRBD resource.

For more detailed instructions regarding initial configuration, please see the chapter 4 of the DRBDUser's Guide.

It is highly recommended that you put your resource configurations in separate resource files that reside in the `/etc/drbd.d` directory, whose name is identical to that of the resource such as **`/etc/drbd.d/gitlab.res`**. Its contents should look similar to this:

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

Copy this configuration to both DRBD nodes. Next it will be necessary to bring the DRBD resources up and online, as they will serve as the backing storage for the virtual machines.

First create the metadata for the DRBD resources. This step must be done on both nodes:

```
# drbdadm create-md gitlab
```

Then load the DRBD module and bring the resource up on both nodes:

```
# drbdadm up gitlab
```

The DRBD resource should now be in the connected state, the *Secondary* role on both nodes, and show a disk state of *Inconsistent* on both nodes. Verify their state by typing the following on either node:

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

At this point we may either begin the initial device synchronization, or, as this is a brand new volume and identical on both nodes already (empty), we can safely skip the initial synchronization with the `--clear-bitmap` option. Run the following command on one node:

```
# drbdadm --clear-bitmap new-current-uuid gitlab/0
```

The **`/0`** at the end of the above command is to specify the volume number of the resource. Even though the above examples do not utilize the multi-volume support in DRBD, it is still required to specify a volume number, 0 being the default.

The DRBD resource should now be in the connected state, the *Secondary* role on both nodes, and show a disk state of

UpToDate on both nodes. Verify their state by typing the following on either node:

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

Now that the resource disk state is no longer inconsistent, promote the resource to *Primary* on the node you wish to use for creation of the virtual domain:

```
# drbdadm primary gitlab
```

3.5. Creating a Filesystem

Once the DRBD resource has been created and initialized, you can create a filesystem on the new block device. This example assumes `xfs` as the filesystem type:

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

You only need to create the filesystem on the *Primary* node. Other filesystems such as `ext4` and `btrfs` may be deployed instead of `xfs`.

3.6. GitLab Data Directory Configuration

After the filesystem has been created it can be mounted temporarily to setup the folder structure under `/mnt/gitlab`.

Create the `/mnt/gitlab` mount point on both nodes:

```
# mkdir /mnt/gitlab
```

Temporarily mount the filesystem on the *Primary* node:

```
# mount /dev/drbd0 /mnt/gitlab
```

Run the following commands on the *Primary* node to move and symlink GitLab's `/etc/gitlab`, `/var/opt/gitlab`, and `/var/log/gitlab` directories:

```
# gitlab-ctl stop

# mv /var/log/gitlab /var/log/gitlab.orig && mkdir -p /mnt/gitlab/var/log/gitlab
# chown git:root /mnt/gitlab/var/log/gitlab
# ln -s /mnt/gitlab/var/log/gitlab  /var/log/gitlab
# rsync -azpv /var/log/gitlab.orig/* /mnt/gitlab/var/log/gitlab

# mv /var/opt/gitlab /var/opt/gitlab.orig && mkdir -p /mnt/gitlab/var/opt/gitlab
# chown root:root  /mnt/gitlab/var/opt/gitlab
# ln -s /mnt/gitlab/var/opt/gitlab  /var/opt/gitlab
# rsync -azpv /var/opt/gitlab.orig/* /mnt/gitlab/var/opt/gitlab

# mv /etc/gitlab /etc/gitlab.orig && mkdir -p /mnt/gitlab/etc/gitlab
# ln -s /mnt/gitlab/etc/gitlab  /etc/gitlab
# rsync -azpv /etc/gitlab.orig/* /mnt/gitlab/etc/gitlab
```

Symlink folders on the *Secondary* node:

```
# gitlab-ctl stop

# mv /var/log/gitlab /var/log/gitlab.orig
# mv /var/opt/gitlab /var/opt/gitlab.orig
# mv /etc/gitlab /etc/gitlab.orig

# ln -s /mnt/gitlab/var/log/gitlab  /var/log/gitlab
# ln -s /mnt/gitlab/var/opt/gitlab  /var/opt/gitlab
# ln -s /mnt/gitlab/etc/gitlab  /etc/gitlab
```

Ensure the GitLab web interface loads on the *Primary* node after starting gitlab:

```
# systemctl start gitlab-runsvdir.service
```

Stop GitLab services and unmount the DRBD backed file system:

```
# systemctl stop gitlab-runsvdir.service
# gitlab-ctl stop

# umount /mnt/gitlab
```

3.7. Configuring Corosync

An excellent example `corosync.conf` file can be found in the appendix of Clusterlab's Clusters from Scratch document. It is highly recommended to take advantage of the support for redundant rings that was introduced in version 1.4. This is done by simply enabling redundant ring support with the `rrp_mode` option, then adding another `interface` sub-section within the totem section.

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: 192.168.122.201
    nodeid: 1
    name: noda-a
  }
  node {
    ring0_addr: 172.16.0.202
    ring1_addr: 192.168.122.202
    nodeid: 2
    name: node-b
  }
}

quorum {
  provider: corosync_votequorum
  two_node: 1
}

logging {
  to_syslog: yes
}
```

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

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

Verify that everything has been started and is working correctly by issuing the following command, you should see similar output to what is below:

```
# crm_mon -rf -n1
Stack: corosync
Current DC: node-a (version 2.0.2.linbit-3.0.el8-744a30d655) - partition with quorum
Last updated: Sun Aug 30 15:20:14 2020
Last change: Sun Aug 30 10:44:21 2020 by hacluster via crmd on node-a

2 nodes configured
0 resources configured

Node node-a: online
Node node-b: online

No inactive resources


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

3.8. Creating a Basic Pacemaker Configuration

In a highly available 2 node cluster using DRBD, you should:

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

To do so, issue the following commands from the CRM shell accessible from the `crm` command on either node (not both):

```
# crm
crm(live)# configure
crm(live)configure# property stonith-enabled="false"
crm(live)configure# property no-quorum-policy="ignore"
crm(live)configure# rsc_defaults resource-stickiness="200"
crm(live)configure# commit
crm(live)configure# exit
bye
```

While STONITH is not entirely necessary as DRBD is a shared-nothing solution. 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 may always contact LINBIT for more information.

Chapter 4. Configuring Cluster Resources

This section assumes you are about to configure a highly available GitLab cluster with the following configuration parameters:

- The DRBD resource backing all GitLab data (database, configuration, repositories, uploads, etc) is named `gitlab`, and it manages the block device `/dev/drbd0`.
- The DRBD block device holds an `xfs` filesystem which is to be mounted to `/mnt/gitlab`.
- The GitLab processes will utilize that filesystem, and the web server will listen on a dedicated cluster IP address, `192.168.122.200`.
- GitLab related processes are controlled by the `gitlab-runsvdir` systemd target. This service starts GitLab's Omnibus interface and required processes.

4.1. Basic Configuration

In order to create the appropriate cluster resources, open the `crm` configuration shell as `root` and issue the following commands:

```
crm(live)# configure
crm(live)configure# primitive p_drbd_gitlab ocf:linbit:drbd \
                    params drbd_resource="gitlab" \
                    op start interval="0s" timeout="240s" \
                    op stop interval="0s" timeout="100s" \
                    op monitor interval="29s" role="Master" \
                    op monitor interval="31s" role="Slave"

crm(live)configure# ms ms_drbd_gitlab p_drbd_gitlab \
                    meta master-max="1" master-node-max="1" \
                         clone-max="2" clone-node-max="1" \
                         notify="true"

crm(live)configure# primitive p_fs_gitlab ocf:heartbeat:Filesystem \
                    params device="/dev/drbd0" \
                           directory="/mnt/gitlab" \
                           fstype="xfs" \
                    op start interval="0" timeout="60s" \
                    op stop interval="0" timeout="60s" \
                    op monitor interval="20" timeout="40s"

crm(live)configure# primitive p_ip_gitlab ocf:heartbeat:IPaddr2 \
                    params ip="192.168.122.200" cidr_netmask="24" \
                    op start interval="0s" timeout="20s" \
                    op stop interval="0s" timeout="20s" \
                    op monitor interval="20s" timeout="20s"

crm(live)configure# primitive p_gitlab systemd:gitlab-runsvdir \
                    op start interval="0s" timeout="120s" \
                    op stop interval="0s" timeout="120s" \
                    op monitor interval="20s" timeout="100s"
```

You **must** set an appropriate shutdown and startup timeout based on your system utilization and expected workload. Failure to do so will cause Pacemaker to prematurely consider operations timed-out and initiate recovery operations.

```
crm(live)configure# group g_gitlab \
                         p_fs_gitlab p_ip_gitlab p_gitlab
crm(live)configure# colocation c_gitlab_on_drbd \
                         inf: g_gitlab ms_drbd_gitlab:Master
crm(live)configure# order o_drbd_before_gitlab \
                         ms_drbd_gitlab:promote g_gitlab:start
crm(live)configure# commit
crm(live)configure# exit
```

Once this configuration has been committed, Pacemaker will:

- Start DRBD on both cluster nodes.
- Select one node for promotion to the DRBD *Primary* role.
- Mount the filesystem, configure the cluster IP address, and start the GitLab server instance on the same node.
- Commence resource monitoring.

4.2. Adding Network Connectivity Monitoring

Finally, Pacemaker may be configured to monitor the upstream network and ensure that GitLab runs only on nodes that have connectivity to clients. In order to do so, pick one or more IP addresses that the cluster node can expect to always be accessible, such as the subnet's default gateway, a core switch, or similar. The following example uses the default gateway with IP address `192.168.122.1`.

Add the `ping` resources as follows:

```
crm(live)# configure
crm(live)configure# primitive p_ping_gw ocf:pacemaker:ping \
                    params host_list="192.168.122.1" \
                           dampen="5s" \
                           multiplier="1000" \
                    op start interval="0s" timeout="60s" \
                    op stop interval="0s" timeout="60s" \
                    op monitor interval="15s" timeout="60s"

crm(live)configure# clone cl_ping p_ping_gw \
                    meta interleave="true"
```

Finally, add a `location` constraint to tie the *Primary* role of your DRBD resource to a node with upstream network connectivity:

```
crm(live)configure# location l_drbd_primary_on_ping ms_drbd_gitlab \
                    rule $role="Master" \
                      -inf: not_defined pingd or pingd lte 0
crm(live)configure# commit
crm(live)configure# exit
```

Once these changes have been committed, Pacemaker will:

- Monitor upstream IP addresses from both cluster nodes.
- Periodically update a *node attribute* for each node with a value corresponding to the number of reachable upstream hosts.

- Move resources away from any node that loses connectivity to upstream IP addresses.

Ensure ICMP echo is not blocked by any firewalls nor IDS filtering on the IP addresses used for ping monitoring. Additional firewall rules or modification of existing allow lists may be required.

Chapter 5. Accessing GitLab

The highly available GitLab server instance is now ready for use. You can now access GitLab through the virtual IP, http://192.168.122.200/.

The default `external_url` value of `https://gitlab.example.com` is defined in `/etc/gitlab/gitlab.rb`. Defining a DNS record or hosts entry on a client machine to resolve to the virtual IP address of `192.168.122.200` is recommended. If the `external_url` needs to be updated, modify the value and run `gitlab-ctl reconfigure` on the *Primary* node.

Chapter 6. Failure Modes

This section highlights specific failure modes and the cluster's corresponding reaction to the events.

6.1. Node Failure

When **one** cluster node suffers an outage, the cluster shifts all resources to the other node. Since DRBD provides a synchronous replica of all GitLab data to the peer node, GitLab will resume function from the peer node.

In this scenario Pacemaker moves the failed node from the *Online* to the *Offline* state, and starts the affected resources on the surviving peer node.

Details of node failure are also explained in chapter 6 of the DRBD User's Guide.

6.2. Storage Subsystem Failure

In case the storage subsystem backing a DRBD-enabled node fails, DRBD transparently *detaches* from its backing device, and continues to serve data over the DRBD replication link from its peer node.

Details of this functionality are explained in chapter 2 of the DRBD User's Guide.

6.3. Service Failure

In case of `gitlab-runsvdir` unexpected shutdown, a segmentation fault, or similar, the `monitor` operation for the `p_gitlab` resource detects the failure and restarts the systemd service.

The `gitlab-runsvdir` systemd service starts the Omnibus GitLab interface. This can be interacted with through the `gitlab-ctl` front end. Omnibus is responsible for running essential sub-processes such as `postgresql`, `nginx`, `grafana`, and others. If one of the sub-processes is killed, Omnibus will attempt to restart it. Running `gitlab-ctl status` will output all processes managed by Omnibus.

6.4. Network Failure

When upstream connectivity is lost, Pacemaker will automatically move resources away from nodes with failed network links. This requires that `ping` monitoring is set up as explained in Adding Network Connectivity Monitoring.

If the DRBD replication link fails, DRBD continues to serve data from the *Primary* node, and re-synchronizes the DRBD resource automatically as soon as network connectivity is restored.

Details of this functionality are explained in chapter 2 of the DRBD User's Guide.

Chapter 7. Alternative HA Configurations

Note that this technical guide merely demonstrates one solution out of many to achieve a highly available GitLab environment.

- *GitLab with NFS* GitLab can support multiple instances for scaling and HA when configured to use NFS for storage. See GitLab's Configuring GitLab for Scaling and High Availability. Highly Available NFS clusters can also be easily created using Pacemaker and DRBD.
- *Omnibus GitLab with External Services* GitLab can be configured in a similar manner as this guide, but with less core services managed by GitLab's Omnibus. For instance, PostgreSQL, NGINX, and Redis can be configured in a non-bundled fashion and defined instead as highly available resources in Pacemaker backed by DRBD.
- *GitLab Geo* A non-CE Premium and Ultimate feature only. GitLab Geo allows a replicated GitLab instance to function as a read-only fully operational instance that can also be promoted in case of disaster.

Chapter 8. Conclusion

Building a highly available GitLab cluster managed by Pacemaker while leveraging DRBD as the backing storage device is a simple solution to ensure the uptime of your GitLab environment in the event of a system failure.

If you have any questions or concerns regarding deployment of the methods outlined in this document with your existing environments, or with your unique system(s) you can always consult with the experts at LINBIT®. See https://www.linbit.com/contact-us/ for contact information.

Chapter 9. Feedback

Feedback regarding any errors, or even just suggestions, or comments regarding this document are encouraged, and very much appreciated. You can contact the author directly using the email address(s) listed on the title page when present, or feedback@linbit.com.

For a public discussion about the concepts mentioned in this white paper, you are invited to subscribe and post to the drbd-user mailing list. Please see the drbd-user mailing list for details.

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
- We're also on IRC. You can find us on Freenode.net in #drbd.
- Clusterlab's Documentation Wiki: https://www.clusterlabs.org/wiki/Documentation
- GitLab Project: https://about.gitlab.com/
- Installing GitLab on CentOS 8: https://about.gitlab.com/install/?version=ce#centos-8
- Configuring GitLab for Scaling and High Availability: https://docs.gitlab.com/12.10/ee/administration/high_availability/gitlab.html

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