Centos 7.6

HA Guide

Using PCS and DRBD as shared storage

Index

Contents

HA Configuration

The following procedure is an outline of the steps required to set up a Pacemaker cluster that includes a XFS file system. After installing and starting the cluster software on all nodes, create the cluster.

1. Creating a High-Availability Cluster with Pacemaker

This chapter describes the procedure for creating a High Availability two-node cluster using `pcs`. After you have created a cluster, you can configure the resources and resource groups that you require. Configuring the cluster provided in this chapter requires that your system include the following components:

1. Two nodes, which will be used to create the cluster. In this example, the nodes used are `master` and `master1`.

2. Dedicated interface card for one to one connection on said nodes (For DRBD).

1.1. Cluster Software Installation

1. On each node in the cluster, install the High Availability Add-On software packages.

```
# yum install pcs pacemaker
```

2. In order to use `pcs` to configure the cluster and communicate among the nodes, you must set a password on each node for the user ID `hacluster`, which is the `pcs` administration account. It is recommended that the password for user `hacluster` be the same on each node.

```
# passwd hacluster
Changing password for user hacluster.
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
```

3. Before the cluster can be configured, the `pcsd` daemon must be started and enabled to boot on start up on each node. This daemon works with the `pcs` command to manage configuration across the nodes in the cluster.
   On each node in the cluster, execute the following commands to start the `pcsd` service and to enable `pcsd` at system start.

```
# systemctl start pcsd.service
# systemctl enable pcsd.service
```

4. Authenticate the `pcs` user `hacluster` for each node in the cluster on the node from which you will be running pcs. The following command authenticates user `hacluster` on `master` for both of the nodes in the example two-node cluster `master`, and `master1`.

```
[root@master ~]# pcs cluster auth master master1
Username: hacluster
Password:
master: Authorized
master1: Authorized
```

1.2. Cluster Creation

This procedure creates a High Availability cluster that consists of the nodes `master` and `master1`.

1. Cluster creation and starting

```
[root@master ~]# pcs cluster setup --name xCAT master master1
Destroying cluster on nodes: xcat, xcatha...
master1: Stopping Cluster (pacemaker)...
master: Stopping Cluster (pacemaker)...
master1: Successfully destroyed cluster
master: Successfully destroyed cluster

Sending 'pacemaker_remote authkey' to 'master', 'master1'
master: successful distribution of the file 'pacemaker_remote authkey'
master1: successful distribution of the file 'pacemaker_remote authkey'
Sending cluster config files to the nodes...
master: Succeeded
master1: Succeeded

Synchronizing pcsd certificates on nodes master, master1...
master: Success
master1: Success
Restarting pcsd on the nodes in order to reload the certificates...
master: Success
master1: Success
```

2. Enable the cluster services to run on each node in the cluster when the node is booted.

```
[root@master ~]# pcs cluster enable --all
[root@master ~]# pcs cluster start --all
```

Check status:

```
[root@master ~]# pcs status
Cluster name: shivay
 Stack: corosync
 Current DC: master (version 1.1.19-8.el7_6.2-c3c624ea3d) - partition WITHOUT quorum
 Last updated: Mon Feb  4 08:13:05 2019
 Last change: Wed Jan 30 08:10:46 2019 by root via crm_resource on master

1. nodes configured
```

2. DRBD Installation and Configuring

2.1. Pre-requisites to setup DRBD configuration.

The DRBD (stands for Distributed Replicated Block Device) is a distributed, flexible and versatile replicated storage solution for Linux. It mirrors the content of block devices such as hard disks, partitions, logical volumes etc. between servers. It involves a copy of data on two storage devices, such that if one fails, the data on the other can be used.

You can think of it somewhat like a network RAID 1 configuration with the disks mirrored across servers. However, it operates in a very different way from RAID and even network RAID.

Originally, DRBD was mainly used in high availability (HA) computer clusters, however, starting with version 9, it can be used to deploy cloud storage solutions.

Below are the mandatory requirement on your cluster, before you start working on DRBD.

- **Dedicated storage device (LVM/Disk/Partition)**
  - Like we have LVM device in brahma and brahma1 → /dev/shared/drbd
- **Dedicated network device in**
  - Like we have eth1 in both the masters i.e brahma and brahma1

2.2. Installing DRBD Packages

DRBD is implemented as a Linux kernel module. It precisely constitutes a driver for a virtual block device, so it's established right near the bottom of a system's I/O stack.

DRBD can be installed from the **ELRepo** or **EPEL** repositories. Let's start by importing the ELRepo package signing key, and enable the repository as shown on both nodes.

```
# rpm --import https://www.elrepo.org/RPM-GPG-KEY-elrepo.org
# rpm -Uvh http://www.elrepo.org/elrepo-release-7.0-3.el7.elrepo.noarch.rpm
```

Then we can install the DRBD kernel module and utilities on both nodes by running:

```
# yum install -y kmod-drbd90 drbd90-utils
```

2.3. Preparing for low-level storage

Now that we have DRBD installed on the two cluster nodes, we must prepare a roughly identically sized storage area on both nodes. This can be a hard drive partition (or a full physical hard drive), a software RAID device, an LVM Logical Volume or any other block device type found on your system.

We already have LVM device ready with us.

```
[root@master ~]# lsblk |grep shared
└─shared-drbd              253:2    0   500G  0 lvm
```

2.4. Configuring DRBD

DRBD's main configuration file is located at /etc/drbd.conf and additional config files can be found in the /etc/drbd.d directory.

To replicate storage, we need to add the necessary configurations in the /etc/drbd.d/global_common.conf file which contains the global and common sections of the DRBD configuration and we can define resources in .res files.

Let's make a backup of the original file on both nodes, then then open a new file for editing (use a text editor of your liking).

```
# mv /etc/drbd.d/global_common.conf /etc/drbd.d/global_common.conf.orig
# vim /etc/drbd.d/global_common.conf
```

Add the following lines in in both files:

```
global {
 usage-count  yes;
}
common {
 net {
  protocol C;
  allow-two-primaries no;
  max-buffers    8000;
  max-epoch-size 8000;
  sndbuf-size 2M;
 }
}
```

Save the file, and then close the editor.

Let's briefly shade more light on the line **protocol C**. DRBD supports three distinct replication modes (thus three degrees of replication synchronicity) which are:

- **Protocol A**: Asynchronous replication protocol; its most often used in long distance replication scenarios.
- **Protocol B**: Semi-synchronous replication protocol aka Memory synchronous protocol.
- **Protocol C**: commonly used for nodes in short distanced networks; it's by far, the most commonly used replication protocol in DRBD setups.

Important: The choice of replication protocol influences two factors of your deployment: **protection** and **latency**. And **throughput**, by contrast, is largely independent of the replication protocol selected.

2.5. Adding a resource

A **resource** is the collective term that refers to all aspects of a particular replicated data set. We will define our resource in a file called **/etc/drbd.d/brahma_drbd.res**.

Add the following content to the file, on both nodes (remember to replace the variables in the content with the actual values for your environment).

Take note of the **hostnames,** we need to specify the network hostname which can be obtained by running the command **uname -n**.

```
resource brahma_drbd {
        disk {
    c-min-rate 80M;
    c-max-rate 720M;
    resync-rate 40M;
  }
        on brahma {
                device /dev/drbd0;
                disk /dev/shared/drbd;
                        meta-disk internal;
                        address 192.168.1.1:7789;
        }
        on brahma1  {
                device /dev/drbd0;
                        disk /dev/shared/drbd;
                        meta-disk internal;
                        address 192.168.1.2:7789;
        }
}
```

Where:

- **on hostname:** the on section states which host the enclosed configuration statements apply to.
- **Brahma_drbd:** is the name of the new resource.
- **device /dev/drbd0:** specifies the new virtual block device managed by DRBD.
- **disk /dev/shared/drbd:** is the block device partition which is the backing device for the DRBD device.
- **meta-disk:** Defines where DRBD stores its metadata. Using Internal means that DRBD stores its meta data on the same physical lower-level device as the actual production data.
- **address:** specifies the IP address and port number of the respective node.

Also note that if the options have equal values on both hosts, you can specify them directly in the resource section.

2.6. Initializing and Enabling Resource

To interact with **DRBD,** we will use the following administration tools which communicate with the kernel module in order to configure and administer DRBD resources:

- **drbdadm:** a high-level administration tool of the DRBD.
- **drbdsetup:** a lower-level administration tool for to attach DRBD devices with their backing block devices, to set up DRBD device pairs to mirror their backing block devices, and to inspect the configuration of running DRBD devices.
- **Drbdmeta:**is the meta data management tool.

After adding all the initial resource configurations, we must bring up the resource on both nodes.

```
# drbdadm create-md test
```

Next, we should enable the **resource**, which will attach the resource with its backing device, then it sets replication parameters, and connects the resource to its peer:

```
# drbdadm up test
```

Now if you run the lsblk command, you will notice that the DRBD device/volume **drbd0** is associated with the backing device **/dev/shared/drbd**:

```
# lsblk
Output:

[root@brahma ~]# lsblk |head -n20
NAME                      MAJ:MIN RM   SIZE RO TYPE  MOUNTPOINT
sda                         8:0    0   1.1T  0 disk
├─sda1                      8:1    0     2M  0 part
├─sda2                      8:2    0     1G  0 part  /boot
└─sda3                      8:3    0   1.1T  0 part
  ├─centos-root           253:0    0 868.3G  0 lvm   /
  ├─centos-swap           253:1    0    48G  0 lvm   [SWAP]
  └─centos-var            253:15   0   200G  0 lvm   /var
sdb                         8:16   0   4.4T  0 disk
└─shared-drbd             253:2    0   500G  0 lvm
  └─drbd0                 147:0    0   500G  0 disk
```

To check the resource status, run the following command (note that the **Inconsistent/Inconsistent** disk state is expected at this point):

```
# drbdadm status brahma_drbd

OR

# drbdsetup status brahma_drbd --verbose --statistics    #for  a more detailed status
```

You can see the state has changed to **Connected**, meaning the two DRBD nodes are communicating properly, and both nodes are in **Secondary** role with **Inconsistent** data.

2.7. Set Primary Resource/Source of Initial Device Synchronization

At this stage, **DRBD** is now ready for operation. We now need to tell it which node should be used as the source of the initial device synchronization.

Run the following command on only one node to start the initial full synchronization:

```
# drbdadm primary --force brahma_drbd
# drbdadm status brahma_drbd
```

Once the synchronization is complete, the status of both disks should be **UpToDate**.

```
[root@brahma ~]# drbdadm status brahma_drbd
brahma_drbd role:Primary
  disk:UpToDate
  brahma1 role:Secondary
    peer-disk:UpToDate
```

2.8. File system Creation

Finally, we need to create filesystem. Remember, we used an empty disk volume, therefore we must create a filesystem on the device, and mount it.

We can create a filesystem on the device with the following command, on the node where we started the initial full synchronization (which has the resource with primary role):

```
#  mkfs -t ext4 /dev/drbd0
```

3. Configuring DRBD, Filesystem in PCS

3.1. Configure the Cluster for the DRBD device

One handy feature pcs has is the ability to queue up several changes into a file and commit those changes all at once. To do this, start by populating the file with the current raw XML configuration from the CIB.

```
# pcs cluster cib tmp-cib.xml

# pcs -f tmp-cib.xml property set stonith-enabled=false

# pcs -f tmp-cib.xml resource defaults migration-threshold=10 resource-stickiness=100 startdelay=45s

# pcs -f tmp-cib.xml resource create vip_xCAT ocf:heartbeat:IPaddr2 \
  cidr_netmask=23 ip=172.10.2.3 \
  op monitor interval=10s timeout=20s start interval=0s timeout=20s stop \
  interval=0s timeout=20s

# pcs -f tmp-cib.xml resource create drbd_fs ocf:linbit:drbd \
  drbd_resource=brahma_drbd \
  op demote interval=0s timeout=90 monitor interval=60s notify interval=0s \
  timeout=90 promote interval=0s timeout=90 reload interval=0s timeout=30 \
  start interval=0s timeout=240 stop interval=0s timeout=100

# pcs -f tmp-cib.xml \
  resource master drbd_fs_clone drbd_fs master-node-max=1 clone-max=2 \
  notify=true master-max=1 clone-node-max=1
```

Using pcs's -f option, make changes to the configuration saved in the tmp-cib.xml file. These changes will not be seen by the cluster until the tmp-cib.xml file is pushed into the live cluster's CIB later.

Here, we have created a cluster resource for the DRBD device, and an additional *clone* resource to allow the resource to run on both nodes at the same time.

3.2. Configure the Cluster for the Filesystem

Now that we have a working DRBD device, we need to mount its filesystem.

In addition to defining the filesystem, we also need to tell the cluster where it can be located (only on the DRBD Primary) and when it is allowed to start (after the Primary was promoted).

We are going to take a shortcut when creating the resource this time. Instead of explicitly saying we want the **ocf:heartbeat:Filesystem** script, we are only going to ask for **Filesystem**. We can do this because we know there is only one resource script named **Filesystem** available to pacemaker, and that pcs is smart enough to fill in the **ocf:heartbeat:** portion for us correctly in the configuration. If there were multiple **Filesystem** scripts from different OCF providers, we would need to specify the exact one we wanted. Once again, we will queue our changes to a file and then push the new configuration to the cluster as the final step.

```
# pcs -f tmp-cib.xml resource create xcatfs ocf:heartbeat:Filesystem \
  device=/dev/drbd0 directory=/xCATdrbd fstype=xfs \
  op monitor interval=20s timeout=40s notify interval=0s timeout=60s start \
  interval=0s timeout=60s stop interval=0s timeout=60s

# pcs -f tmp-cib.xml constraint colocation add xcatfs with master drbd_fs_clone

# pcs -f tmp-cib.xml constraint order promote drbd_fs_clone then xcatfs


# pcs -f tmp-cib.xml constraint location drbd_fs_clone prefers brahma=100

# pcs -f tmp-cib.xml constraint colocation add master drbd_fs_clone with vip_xCAT
```

Push the configuration

```
# pcs cluster cib-push tmp-cib.xml -config
```

Check the status using pcs status.

3.3. Configuring xCAT in PCS

In order to configure xCAT in PCS. We need total 7 directories to be on central location i.e in DRBD (/xCATdrbd)

1. /etc/xcat → xCAT database i.e sqlite
2. /opt/xcat → xCAT commands i.e lsdef, chdef
3. /install → Compute images location
4. /tftpboot → tftp config
5. /root/.xcat → xCAT keys
6. /etc/dhcp → DHCP configuration i.e dhcpd.conf
7. /var/lib/dhcpd → DHCP lease file

We have to take backup of all above directories from any one of two nodes (node which has some configuration i.e site table, network etc.)

Now move this directories to /xCATdrbd which is a shared storage pertition.

1. /etc/xcat → /xCATdrbd/etc_xcat
2. /opt/xcat → /xCATdrbd/opt_xcat
3. /install → /xCATdrbd/install_xcat
4. /tftpboot → /xCATdrbd/xcat_tftpboot
5. /root/.xcat → /xCATdrbd/keys_xcat
6. /etc/dhcp → /xCATdrbd/dhcp_xcat
7. /var/lib/dhcpd → /xCATdrbd/xcat_var_lib_dhcpd

```
# pcs resource create tftpboot_symlink ocf:heartbeat:symlink link=/tftpboot
target=/xCATdrbd/xcat_tftpboot

# pcs resource create etc_xcat_symlink ocf:heartbeat:symlink link=/etc/xcat
target=/xCATdrbd/etc_xcat

# pcs resource create install_xcat_symlink ocf:heartbeat:symlink link=/install
target=/xCATdrbd/install_xcat \

# pcs resource create keys_xcat_symlink ocf:heartbeat:symlink link=/root/.xcat
target=/xCATdrbd/keys_xcat \

# pcs resource create opt_xcat_symlink ocf:heartbeat:symlink link=/opt/xcat
target=/xCATdrbd/opt_xcat

# pcs resource create etcdhcp_symlink ocf:heartbeat:symlink  link=/etc/dhcp
target=/xCATdrbd/dhcp_xcat

# pcs resource create dhcplease_symlink ocf:heartbeat:symlink link=/var/lib/dhcpd
target=/xCATdrbd/xcat_var_lib_dhcpd \
```

3.4. Adding service as a resource in PCS

We are adding only one service which is must in a failover

```
pcs resource create ser_dhcp systemd:dhcpd \
  op monitor interval=30s start interval=0s timeout=100 stop interval=0s \
  timeout=100
```

We can add more service as required. But more the services, more the complexity.

3.5. Setting constraints in PCS for resources

There are three types of resources

Constraints are rules that place restrictions on the order in which resources or resource groups may be started, or the nodes on which they may run. Constraints are important for managing complex resource groups or sets of resource groups, which depend upon one another or which may interfere with each other.

There are three main types of constraints:

- **Order constraints**, which control the order in which resources or resource groups are started and stopped.
- **Location constraints**, which control the nodes on which resources or resource groups may run.
- **Colocation constraints**, which control whether two resources or resource groups may run on the same node.

First we will create resource group so that constraint setting will go easier:

```
# pcs resource group add Symblinks tftpboot_symlink etc_xcat_symlink install_xcat_symlink
keys_xcat_symlink opt_xcat_symlink etcdhcp_symlink dhcplease_symlink


# pcs resource group add Service ser_dhcp
```

```
# pcs constraint location vip_xCAT prefers brahma1=50

# pcs constraint location vip_xCAT prefers brahma=100

# pcs constraint colocation add Symblinks with xcatfs

# pcs constraint colocation add Service with xcatfs

# pcs constraint order set vip_xCAT drbd_fs_clone xcatfs Symblinks Service
```

References

https://clusterlabs.org/pacemaker/doc/en-US/Pacemaker/1.1/html/Clusters_from_Scratch/_configure_the_cluster_for_the_filesystem.html