“dycom-hacluster”

based on

Tutorial / Manual

2-Node HA (High Availability) Cluster
w/ Linux Ubuntu Server 16.04 LTS on
Physical Machines

DRBD(8), Pacemaker 2.4 (CRM), Corosync,
Samba*

*in Active/Passive Redundancy Configuration

2019.01.26

Introduction

This document serves the purpose of summarizing the configuration implemented for the 2-node HA cluster as requested by *Dycom Ltd.*, and is entirely based on the tutorial/manual entitled *2-Node HA (High Availability) Cluster w/ Linux Ubuntu Server 16.04 LTS on Physical Machines*. It is assumed that the reader (eg. system admin) can and will reference this tutorial/manual in conjunction with the content following.

Requirements

In this project we assume the following setup:

- The HA cluster will consist of 2 nodes, each with 1 no. 3TB hard drive attached either via USB or directly on motherboard, and 2 network interface cards (NICs).
- The 3TB hard drive (block device) will have one ext4 partition of 3TB size – the exact same on both nodes
- DRBD will be used for real-time data replication from local node on ext4 file system of backing block device – via DRBD device */dev/drbd0* - mounted to the file system at */dev/drbd/3TB*, to same location on LAN connected node (NIC-to-NIC via cross-over Ethernet)
    - 3TB ext4 partition mounted to DRBD device at */dev/drbd/3TB*
        - Partition *1* from physical 3TB block device at eg. */dev/sdb*
- Pacemaker/Corosync will be used for cluster resource management (CRM) and messaging layer respectively
- Samba fileserver in active/passive redundancy configuration
    - We assume samba has been installed and identically setup on all nodes in the cluster, and that it has been disabled
- Virtual IP
    - this is a static IP address that can be moved (aka *migrate*) between nodes
    - when the virtual IP has migrated to a node, that node will then be accessible via either the interface to its NIC, or the virtual IP
    - usual clients (eg. SMB or VPN clients) should only use <u>this</u> virtual IP address to contact the service running on the node - <u>not</u> the actual IP address assigned to node (eg. /etc/hosts) either manually or via DHCP
    - Care should be taken at the static IP chosen to be the virtual IP. DHCP services should be setup to delegate IP addresses to net devices using a range that does not include the static IP chosen to be the virtual IP.

In this 2-node HA cluster, all file transactions committed to 3TB partition mounted on the DRBD device on the <u>primary</u> node, must be replicated in real time to the same size partition mounted on the DRBD device on the <u>secondary</u> node.

Change Host Name

First we change the old (current) host name on each node to something more applicable for the cluster setup:

Node 1 (either of the nodes):

```
sudo nano /etc/hostname

     dycom-slave          #new host name
```

Node 2:

```
sudo nano /etc/hostname

     dycom-slave          #new host name
```

Note, */etc/hostname* on each node should contain <u>one</u> host name only.

Now we make the following amendments in the */etc/hosts* file of <u>each</u> node:

Node 1 ("dycom-master"):

```
sudo nano /etc/hosts

     127.0.0.1       localhost
     127.0.1.1       dycom-master    #new host name
     ...
```

Node 2 ("dycom-slave"):

```
sudo nano /etc/hosts

     127.0.0.1       localhost
     127.0.1.1       dycom-slave     #new host name
     ...
```

Restart both nodes so that the new host names take effect.

Basic Setup

- `/etc/network/interfaces`
    - Ex1. shows how to set up having only 1 network interface card (NIC) per node
    - Ex2. Shows how to set up when you have a 2nd NIC in each node
        - 2nd NIC creates **isolated** network – let's call this the DRBD network - for both communication between nodes, and also for DRBD data to be mirrored
        - Nodes are to be connected via 2nd NIC either via cross-over Ethernet cable directly, or via standard Ethernet cable via standalone switch.
        - **NOTE**: the isolated network implies no additional security is required as long as the switch is not connected to any other node outside.

Ex1: Node with one (1) NIC only – static IP assigned via DHCP

Note: not implemented in "dycom-hacluster" – included for reference only

```
# This file describes the network interfaces available on your
# system and how to activate them. For more information, see
# interfaces(5).
source /etc/network/interfaces.d/*

# The loopback network interface                  *1*
auto lo
iface lo inet loopback

# The primary network interface                   *2*
auto enp4s0
iface enp4s0 inet dhcp

# This is an autoconfigured IPv6 interface
iface enp4s0 inet6 auto
```

Where:

- *1* is the loop back network – ie. a direct reference to the host itself
- *2* refers to the only network interface card (enp4s0) installed on the machine, and that the IP address is assigned via DHCP from a router.
    - Note, even though this configuration states a dynamic IP configuration; it is vital that the IP assigned by the router (via DHCP) to remain STATIC at all times.
        - If you do NOT intend to make use of DHCP at all and would rather assign a static IP, you could follow the syntax shown for the 2nd NIC in example 2 below. Again, take note that the IP should fall outside of the range used by the DHCP service (if enabled on the router) so that duplicate assignments can be avoided.
    - DHCP also implies that the network address of these solitary NICs would be DHCP assigned
        - The *Network Address* is a virtual IP address reserved to differentiate the network itself from any other network
            - The network address is usually the same as the that of a NIC, but with the host address equal to 0.

- Eg.: if the IP assigned to a NIC – either statically or dynamically - is 192.168.178.30, then the network that NIC is connected to is reserved the IP address of 192.168.178.0

Ex2: Node with two (2) NICs – static IP assigned manually to 2nd NIC
Note: this is version implemented on "dycom-hacluster"

The difference from ex1 is shown in the lines in red.

```
# This file describes the network interfaces available on your
# system and how to activate them. For more information, see
# interfaces(5).
source /etc/network/interfaces.d/*

# The loopback network interface                            *1*
auto lo
iface lo inet loopback

# The primary network interface                             *2*
auto enp4s0
iface enp4s0 inet dhcp

# The secondary network interface                           *3*
auto enp2s0
iface enp2s0 inet static
address     10.1.1.1
netmask     255.255.255.0

# This is an auto configured IPv6 interface
iface enp4s0 inet6 auto
iface enp2s0 inet6 auto                                     *4*
```

Where:

- *1* see ex1 description
- *2* see ex1 description
- *3* is the STATIC IP address setup
  - This address can be any IP address, technically, as long as it is unlikely to clash with other IP addresses assigned to other nodes on the network. For this reason we use a reserved IP range such as 10.x.x.x
  - Note that in the example above, 10.1.1.1 is the IP address for the master node. Change this address to, say, 10.1.1.2 on the slave node.
  - The network mask of 255.255.255.0 indicates a /24 network
    - ie.: leaving the last 8 bits for host addresses (1 – 254)
    - implies that the 2nd NIC to have a network address of 10.1.1.0
- *4* refers to the version6 IP interface of the 2nd network interface card (enp3s1) installed on the machine, and that the IP address to be calculated automatically (assumably from IPv4 address).

- `/etc/hosts`
  - To resolve domain-name to host-name to IP address

- Same for both master/primary and slave/secondary (ie. all) nodes in the cluster, except for the host name used for loopback 127.0.1.1; where the differentiation should reflect the host-name given to each node.

Ex1: One (1) NIC setup – static IPs assigned via DHCP
Note: not implemented in "dycom-hacluster" – included for reference only

```
127.0.0.1       localhost
127.0.1.1       dycom-master     *1* optional (can comment out)

# The following lines are desirable for IPv6 capable hosts
::1     localhost ip6-localhost ip6-loopback
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters

# ADDED FOR 2-NODE FAIL-OVER CLUSTER (DRBD-based)
# The following lines are required to resolve node names.
# Note the use of STATIC IPs (so either
# /etc/network/interfaces or router has to be setup for
# static IPs)
#
# IP              TLDN (randomly made up)     HOST NAME
#
192.168.178.30   dycom-master.co.nz          dycom-master      *2*
192.168.178.54   dycom-slave.co.nz           dycom-slave       *3*
192.168.178.201  dycom-hacluster.co.nz       dycom-hacluster   *4*
```

Where:

- *1*OPTIONAL: is the host name of the node in question
  - Dependent on the node this file is located on – and must equal the relevant host name found in either *2* or *3*
  - This is the only difference between /etc/hosts from the primary (master) and secondary (slave) nodes

- *2* = IP to Top-Level-Domain-Name (TLDN) & Host Name resolution
  - IPs stated must reflect those in /etc/network/interfaces

- *3* = see *2*
- *4* = a highly available IP that can migrate (move) between the nodes; where usual clients (such as SMB clients and VPN clients) should only use this IP to contact the services.

  The choice in IP should also reflect the subnet of the LAN shared with clients such as Samba users (see *Samba Server: Install and Configure* section later).

Ex2: Two (2) NIC setup – static IPs manually assigned
Note: this is version implemented on "dycom-hacluster"

```
127.0.0.1       localhost
```

```
127.0.1.1       dycom-master                                      *1*

# The following lines are desirable for IPv6 capable hosts
::1     localhost ip6-localhost ip6-loopback
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters

# ADDED FOR 2-NODE FAIL-OVER CLUSTER (DRBD-based)
# The following lines are required to resolve node names.
# Note the use of STATIC IPs (so either
# /etc/network/interfaces or router has to be setup for
# static IPs)
#
# IP       TLDN (randomly made up)     HOST NAME
#
10.1.1.1   dycom-master.co.nz          dycom-master               *2*
10.1.1.2   dycom-slave.co.nz           dycom-slave                *3*
192.168.178.201 dycom-hacluster.co.nz  dycom-hacluster            *4*
```

Where:

- See *1*, *2*, *3* and *4* from ex1 for descriptions
- IP addresses must reflect those of the 2nd NIC give in `/etc/network/interfaces`

Command Syntax Reference

The syntax used to describe any command henceforth will include some simple tokens:

- <> denotes a required string, eg.: name or id,
- `[]` means that the construct is optional,
- the ellipsis or parantheses `(...)` signifies that the previous construct may be repeated,
- `|` means pick one of many,
- the double colon `::` signifies an attempt to elaborate on a command, property or attribute. It is not part of an actual command, and is purely included for clarification.
    - **Note to self:** this seems to be the case when looking at the examples in the underlying source article used for this manual. Has to be confirmed still.
- and the rest are literals (*strings*, `:`, =).

DRBD Network (Distributed Resource Block Device)

Install Software

- `sudo apt-get update`
- `sudo apt-get install ntp drbd8-utils -y`

```
            # ntp = network time protocol,
            # drbd8-utils for DRBD kernel module (driver
```

Internal Meta Data: Creating a New Partition on Block Device

On both nodes - create a partition or logical volume on the disk (block device) where you will place the user/production data you want replicated - **but do not format it**. It is important to note that the size of the partition (or logical volume) on both nodes must be exactly the same size. These partitions (on block devices) are referred to as DRBD "backing block-devices", so called because they are mounted to the DRBD device (eg. */dev/drbd0*), from where the DRBD device really acts as an interface to the partition it is associated with.

Eg.:

```
sudo fdisk /dev/sdb
```

(where b might differ independently, depending on the device numbering)

After you've created the partition (or logical volume) - zero out your partition on both nodes, ie. clear all data on these partitions (optional, but recommended):

Eg.:

```
sudo dd if=/dev/zero of=/dev/sdb1
```

(where b1refers to the partition you created in the previous step)

Configure Nodes

Step 1: Create the configuration file for your DRBD resource(s)

Note: Each DRBD *device* (explained next) will be managing a single volume (ie. logical volume or partition) in a DRBD *resource*. This way you can configure for more than one drbd device (eg. *drbd0*, *drbd1*, etc.), and more than one drbd resource (eg. *resource r0 {…}, resource r1 {…}*, etc.).

The DRBD device is usually named /dev/drbdX, where X corresponds to the resource number in */etc/drbd.conf*.

We start by editing the DRBD configuration file, eg.:

- `sudo nano /etc/drbd.conf`

```
#include "drbd.d/global_common.conf";
#include "drbd.d/*.res";

global {
      usage-count no;
}

common {
      # protocol to use; C is the the safest variant
      protocol C;                                   # *1*
}

#Resource '0' (ie. block device or volume to be replicated,
#starting with no. 0).  Each resource is managed by a DRBD device
#(a virtual block device).
# An associated DRBD device is usually named /dev/drbdX,
# where X corresponds to the resource number.
resource r0 {
      # name of the allocated DRBD device
      device   /dev/drbd0;                          # *4*


      # where to store DRBD metadata; here it's on the underlying
      # “backing” block device itself
      meta-disk       internal;                     # *6*

      startup {
            # timeout (in seconds) for the connection on startup
            wfc-timeout     15;

            # timeout (in seconds) for the connection on startup
            # after detection of data inconsistencies ("degraded
            # mode")
            degr-wfc-timeout        60;
      }

      syncer {
            # maximum bandwidth to use for this resource (M = MB/s)
            # 100M = 800Mbps; you may try and up this to 120M
            rate    100M;
      }
```

```
        connection-mesh {                           # *2*
              hosts dycom-master dycom-slave;
        }

        on dycom-master {                           # *3*
              ### options for master-server ###

              # DRBD underlying "backing" block device
              # This will be the block device from which file
              # operations will be replicated
              disk     /dev/sdb1;                   # *5*

              # address and port to use for the synchronisation
              # here we use the heartbeat or Corosyncnetwork interface
              address        10.1.1.1:7788;
              #address         192.168.178.30:7788;

              # OPTIONAL:  to maintain a unique node identification
              # irrespective of cluster role delineation.  Each node
              # must have a unique ID.
              # node-id   0;
        }

        on dycom-slave {                            # *3*
              disk     /dev/sdb1;                   # *5*

              address        10.1.1.2:7788;
              #address         192.168.178.54:7788;
              #node-id    1;
        }

        handlers  {                                 # *7*
              split-brain "/usr/lib/drbd/notify-split-brain.sh root"
        }

        #automatic recovery policies
        #  (ref. https://docs.linbit.com/docs/users-guide-8.4/#s-configure-split-brain-behavior)
        #
        net  {                                      # *8*
              after-sb-0pri discard-least-changes;
              after-sb-1pri discard-secondary;
              after-sb-2pri disconnect;

              #native max-buffers setting:  increase sync rate
              #  (ref. https://www.halizard.com/forum/software-
              #  support/310-drbd-sync-rates)
              max-buffers 100000;
        }
}
```

Most important points:

1 The specified protocol to be used for this connection.
For protocol C, a *write* operation is considered to be complete when it has reached all disks, be they local or remote.

2 OPTIONAL: Defines all nodes of a mesh.
The hosts parameter contains all host names that share the same DRBD setup.

3 Node names can be anything– as long as its different for each node.
Contains the IP address and OPTIONAL unique identifier for each node.

4 The DRBD device that applications are supposed to access.

5	The lower-level or "backing" block device used by DRBD to store the actual data.
6	Where the meta data format is stored. Using *internal*, the metadata is stored together with the user data on the same device. Using *external*, it is not, and should include the volume/location where it can be found.
7	Split-brain notification: see section *Split-Brain*
8	Split-brain automatic recovery policies: see section *Split-Brain*
9	IP address of interface and port to use for synchronisation

Step 2: Load the DRBD Kernel Module

To load the DRBD driver module into the kernel, we execute:

- `sudo modprobe drbd`

Step 3: Set Firewall Rules

Add the firewall rules (TCP ports) to allow traffic on both nodes (don't open port on router for public access!!!)

- `sudo ufw allow 7788/tcp`

Step 4: Create & Enable DRBD Resource

On both nodes, create DRBD's metadata on the resource,

- `sudo drbdadm create-md <resource>`

eg.:

```
sudo drbdadm create-md r0   #create metadata space on
                           #for DRBD resource 'r0'
                           #(on r0's backing block device)
```

eg. output:

```
initializing activity log
NOT initializing bitmap
Writing meta data...
New drbdmeta data block successfully created.
```

Now <u>enable</u> (or "bring up") the drbd resource on a host:

- `sudo drbdadm up <resource>`

Eg.:

-
  ```
  sudo drbdadm up r0          #start the drbd service for
                             #resource r0 on current host
  ```

This should also start up the DRBD service, but if it doesn't:

- `sudo systemctl start drbd`   `#TBC takes default configured resource (eg. r0)`

side note:

drbd is the start and stop script for DRBD, and is located at /etc/init.d/drbd In order to use /etc/init.d/drbd you must define a resource, a host, and any other configuration options in the drbd configuration file. See /etc/drbd.conf for details. If resource is omitted, then all of the resources listed in the config file are configured.

Should you require to <u>disable</u> a DRBD resource on a host:

- `sudo drbdadm down <resource>`

Check the output of `cat /proc/drbd` to observe the DRBD status

side note:

/proc is a virtual and pseudo file-system which contains information about running processes with a particular process-ID, aka PID.

Both primary and secondary nodes are set (or demoted) to 'secondary' nodes initially; so you should see your device is:

Connected, Secondary/Secondary, and Inconsistent/Inconsistent

eg. output:

```
version: 8.4.5 (api:1/proto:86-101)
srcversion: 611D9EEFB9C11D2BC709D07
```

```
0:      cs:Connectedro:Secondary/Secondaryds:Inconsistent/Inconsistent C r-----
        ns:0 nr:0 dw:0 dr:0 al:0 bm:0 lo:0 pe:0 ua:0 ap:0 ep:1 wo:f oos:1048508
```

If any of these things are NOT true, STOP, something isn't right. Back track and start over; do NOT continue until this the above output.

Looking at the scenario example stated at the start, an output for command `sudo lsblk` **on both master and slave nodes** – focusing on the 1GB partition on a 28.9GB block device - would look something like this:

```
NAME      MAJ:MIN   RM   SIZE  RO   TYPE  MOUNTPOINT
sda   8:0       0    149G      0    disk
├─sda1     8:1      0    18.6G      0    part /
├─sda2     8:2      0    488M       0    part /boot
├─sda3     8:3      0    5.6G       0    part [SWAP]
├─sda4     8:4      0    1K         0    part
└─sda5     8:5      0    4.9G       0    part /home
sdb   8:16      1    28.9G     0    disk
└─sdb1     8:17     1    1G         0    part
sr0   11:0      1    1024M     0    rom
drbd0147:0      0    1024M     1    disk
```

Step 5: Define Master/Slave Roles

Now, select any node (NOT BOTH), and define it to be the Primary (Master) node with either:

- `sudo drbdadm -- --overwrite-data-of-peer primary all`
  - preceding '--' is not a typo!!!!
  - The *primary* property signifies this node to be the primary node
  - You can replace option *all* with the name of the resource given in */etc/drbd.conf*.

- `sudo drbdadm -- --overwrite-data-of-peer primary r0`

- `sudo drbdadm -- --overwrite-data-of-peer primary r0/0`
  - where r0/0 implies the 0th instance of resource r0

- `sudo drbdadm primary --force <resource>`

  eg.:

  `sudo drbdadm primary --force r0`

NOTEs:

DRBD will not let you set a node to primary role with *Inconsistent* data.

The command above will cause DRBD to sync (and **overwrite**) any data for this resource to the peer node.

Synching will occur on first connection of the nodes to the DRBD network, even when the file system on the block device appears to have no user data. **This procedure can take a very long time on large volumes.**

To pause/resume synchronication (on either node) you may issue this command:

- `sudo drbdadm <pause-sync | resume-sync> <resource>`

To view the synchronisation (on either node) issue this command:

- `watch -n 1.0 'sudo cat /proc/drbd'`

  eg.:

  PRIMARY (master) node output:

```
version: 8.4.5 (api:1/proto:86-101)
srcversion: 611D9EEFB9C11D2BC709D07
 0:     cs:Connectedro:Primary/Secondaryds:UpToDate/UpToDate C r-----
        ns:0 nr:0 dw:0 dr:640 al:0 bm:0 lo:0 pe:0 ua:0 ap:0 ep:1 wo:f oos:0
```

  SECONDARY (slave) node output:

```
version: 8.4.5 (api:1/proto:86-101)
srcversion: 611D9EEFB9C11D2BC709D07
 0:     cs:Connectedro:Secondar/Primaryds:UpToDate/UpToDate C r-----
        ns:0 nr:1081932 dw:1081932 dr:0 al:0 bm:0 lo:0 pe:0 ua:0 ap:0 ep:1 wo:f oos:0
```

where:

`cs` (connection state)	= Status of the network connection.
`ro` (roles)	= Roles of the nodes. The role of the local node is displayed first, followed by the role of the partner node shown after the slash.
`ds` (disk states)	= State of the hard disks. Prior to the slash the state of the local node is displayed, after the slash the state of the hard disk of the partner node is shown.
`ns` (network send)	= Volume of net data sent to the partner via the network connection; in Kibyte.
`nr` (network receive)	= Volume of net data received by the partner via the network connection; in Kibyte.
`dw` (disk write)	= Net data written on local hard disk; in Kibyte.
`dr` (disk read)	= Net data read from local hard disk; in Kibyte.
`al` (activity log)	= Number of updates of the activity log area of the meta data.
`bm` (bit map)	= Number of updates of the bitmap area of the meta data.
`lo` (local count)	= Number of open requests to the local I/O sub-system issued by DRBD.
`pe` (pending)	= Number of requests sent to the partner, but that have not yet been answered by the latter.
`ua` (unacknowledged)	= Number of requests received by the partner via the network connection, but that have not yet been answered.
`ap` (application pending)	= Number of block I/O requests forwarded to DRBD, but not yet answered by DRBD.
`ep` (epochs)	= Number of epoch objects. Usually 1. Might increase under I/O load when using either the barrier or the none write ordering method. Since 8.2.7.
`wo` (write order)	= Currently used write ordering method: b (barrier), f (flush), d (drain) or n (none). Since 8.2.7.
`oos` (out of sync) =	Amount of storage currently out of sync; in Kibibytes. Since 8.2.6.

Step 6: Mount the DRBD Device To The File System

Once the master role has been defined, create the file system (FS) on the DRBD device (`mkfs.ext4 /dev/drbd0`) on the same node you marked as the primary, then mount the DRBD device (ie. effectively the FS on the backing block device) to the file system on Linux (`/mnt/drbd/3TB`):

```
sudo mkfs.ext4 /dev/drbd0
sudo mkdir -p /mnt/drbd/3TB
sudo mount /dev/drbd0 /mnt/drbd/3TB
```

NOTE:

do NOT create a file-system on the secondary node – only the primary node should write data to the DRBD backing block device. This created file system (on the backing block device), will of course, be replicated to the secondary node.

IMPORTANT POINT: You'll use the DRBD device *(/dev/drbd0*) just like you would have used the block device (eg. */dev/sdb*); the DRBD device acts as interface between the file system and backing block device. Do NOT directly attempt access of the backing block device (eg. */dev/sda* or */dev/sdb*) ever again unless you're confident in what you're doing. Touching a DRBD device's backing block device can introduce inconsistencies that DRBD will be unaware of (until you run a verify or otherwise overwrite the block).

Revisiting the requirement stated at the start - the output for command `sudo lsblk` on the slave node remains unchanged, but on the master node the DRBD device (*drbd0* or */dev/drbd0*) – which effectively interfaces to the backing block device - has now been mapped to file system on Linux at mount point */mnt/drbd/3TB*.
Focusing on the partition on the physical backing block device – the command output would look something like this:

```
NAME      MAJ:MIN    RM   SIZE RO   TYPE MOUNTPOINT
sda  8:0        0    149G      0    disk
├─sda1    8:1        0    18.6G     0    part /
├─sda2    8:2        0    488M      0    part /boot
├─sda3    8:3        0    5.6G      0    part [SWAP]
├─sda4    8:4        0    1K        0    part
└─sda5    8:5        0    4.9G      0    part /home
sdb  8:16       1    28.9G     0    disk
└─sdb1    8:17       1    1G        0    part
sr0  11:0       1    1024M     0    rom
drbd0147:0      0    2.7T      0    disk /mnt/drbd/3TB
```

Testing Manual Fail-Over *

First Make Sure:

- all cluster nodes have the DRBD service running and configured to start at startup
  - check with: `sudo systemctl status drbd`
  - start with: `sudo systemctl start drbd`

- all cluster nodes have the DRBD device is enabled
  - check with: `sudo cat /proc/drbd`
  - enable with: `sudo drbdadm up <drbd-resource>`
    - eg. `sudo drbdadm up r0`

- (only) one node is set as primary role
  - if in doubt - typically the one last updated
    - `sudo drbdadm primary <drbd-resource>`
    - eg. `sudo drbdadm primary <r0>`

- the primary node (selected in previous step) has the the DRBD device mounted to the file system
  - check with: `sudo lsblk`
  - mount with: `sudo mount <drbd-device> <fs-drbd-device-mount-point>`
    - eg: `sudo mount  /dev/drbd0  /mnt/drbd/1G-USB`

Copy Some Stuff

To test that the data is actually syncing between the hosts, copy some files on the primary (master) node, to the DRBD device (ie. Backing block device):

- `sudo cp -r /etc/default <fs-drbd-device-mountpoint>`
  - eg.: `sudo cp -r /etc/default /mnt/drbd/1G-USB`

Perform Basic Fail-Over:

1. Unmount the DRBD device from the file system on the primary node:
   - `sudo umount <fs-drbd-device-mount-point>`
   - eg.:
     - `sudo umount /mnt/drbd/1G-USB`
   - you should not be able to view the content of the backing block device now

1. Demote the primary node to the secondary role:
   - On the primary server, run:
     - `sudo drbdadm secondary <resource>`
       - eg.: `sudo drbdadm secondary r0`

2. Now promote the secondary node to the primary role:
   - `sudo drbdadm primary <resource>`
     - eg.: `sudo drbdadm primary r0`

3. Lastly, on the secondary node, mount the DRBD device to the file system:
   - `sudo mount <drbd-device> <fs-drbd-device-mount-point>`
   - eg.:
     - `sudo mount /dev/drbd0 /mnt/drbd/1G-USB`
   - Using `ls` you should see the new data copied from the former primary host.

*for more frequently used commands, see the DRBD quick or full reference guide.

Split-Brain

Split-Brain (SB) Email Notifications

(ref. https://docs.linbit.com/docs/users-guide-8.4/#s-configure-split-brain-behavior)

Option 1: use cluster resource `ocf:heartbeat:MailTo`

- OCF resource to be included in the CRM configuration
- Implemented in this project

Option 2: use split-brain handler provided by distribution

- Linux supplied script(s) configured in *handlers* section in */etc/drbd.conf*
    - No resource to be added or managed in CRM
    - Automatically invoked on next split-brain
- DRBD comes pre-packaged with a split-brain handler at */usr/lib/drbd/notify-split-brain.sh*.
- Eg.:

```
...
handlers {
  split-brain "/usr/lib/drbd/notify-split-brain.sh";
}
...
```

- **Not** implemented for this cluster due to unresolved issues
    - As with *option 1*: including a *handlers* section in *drbd.conf* causes the DRBD to fail

Option 3: use split-brain handler from custom scripts

- Custom written script(s) configured in *handlers* section in */etc/drbd.conf*
    - No resource to be added or managed in CRM
    - Automatically invoked on next split-brain
- Eg.:

```
...
handlers {
  split-brain "echo Split-Brain in DRBD | mail –s 'DRBD Alert' root";
}
...
```

- **Not** implemented for this cluster due to unresolved issues
    - As with *option 2*: including a *handlers* section causes the DRBD to fail

Option 3 uses a local, send-only SMTP server (e.g. Postfix) – which requires no dedicated email or 3rd party SMTP server. A full setup tutorial can be found in another of our tutorials, or online at either DigitalOcean or StackOverflow.

Once the Postfix MTA (mail transfer agent) has been installed and tested, we can follow the same method explained in *Option 2*, but this time we'll introduce piping a custom string value to the `mail` script of our Postfix MTA. *root*, can be forwarded in the */etc/aliases* file to any email address.

Note to self:
Google's Gmail requires the use of SPF records for authentication before accepting emails.

Split-Brain Confirmation for Users

When DRBD detects a split-brain scenario, it immediately stops the replication connection.
At this point, at least one node in the cluster will always have the DRBD resource (eg. *r0*) in a *StandAlone* connection state. If both nodes detected the split-brain at the same time, the other 2nd node might either be in the *StandAlone* state as well, or in *WFConnection* state. The latter occurs if the peer brought down the connection before the other node registered the split-brain.

Part of the problem with SB is for the user to recognise when the issue occurs – see Tutorial for full details:

Manual Split-Brain Resolve Options

Refer to the tutorial for full details.

Automatic Split-Brain Recovery Policies:

DRBD provides a few resolution algorithms for automatic recovery based on the number of nodes in the *primary* role at the time split-brain is detected. In that vein, DRBD will examine **key-words** listed in the cluster resource's *net* configuration section in */etc/drbd.conf*. The following configuration had been implemented

(full descriptions available in the Tutorial/Manual)

```
. . .
#automatic recovery policies
net {
     after-sb-0pri discard-least-changes;
     after-sb-1pri discard-secondary;
     after-sb-2pri disconnect;
     . . .
  }
. . .
```

Clusters: Introducing Pacemaker & Corosync

Samba Server

Install and Configure

The installation and configuration of Samba does not fall within the scope of this tutorial, but a detailed approach can be followed in another of our tutorials called *Ubuntu Server 16.04 LTS Installation & Configuration*, section *8.1 File Sharing with SMB & Samba*.

In this project, Samba is set up in an *active/passive* redundancy configuration – which essentially entails making sure that Samba will only ever run on one node at a time - as opposed to running on both servers at the same time in an active/active configuration. All of this is managed by the CRM, of which you'll see later in the *Sample CIB Configuration* section.

Assuming that you've configured the respective */etc/hosts* and */etc/network/interfaces* files with a virtual IP of *192.168.178.201* on a */24* netmask, and that it resolves to host name *dycom-hacluster.co.nz* - the following lines to the *[Global]* section in the */etc/samba/smb.conf* file could look like this:

```
[global]
...
# The following is to make sure nodes in the High Available (HA) cluster
# use the same netbios name, so that windows shares stay available after
# the virtual IP migrates (moves) to another node:
  netbios name = <dycom-hacluster.co.nz>
  interfaces = 192.168.178.201/24
  bind interfaces only = true
...
```

For shares to Windows machines, include the following lines in */etc/samba/smb.conf*:

```
##===================DYCOM SETUP FROM THIS POINT FORWARD===================
#
##UPDATED FOR 2NODE HA CLUSTER
[fileshare]
  comment = Common space to share files
  #path = /mnt/storage/shared
  path = /mnt/drbd/3TB/storage/shared
  read only = no
  create mask = 0777
  force create mode = 0777
  directory mask = 0777
  force directory mode = 0777

  #force user - see man.smbf
  ;force user = a_user

  #force group: *** uncomment the command below if you struggle with
  #permissions (see man smb.conf) ***
  # This specifies a UNIX group name that will be assigned as the
  # default primary group for all users connecting to this service. This
  # is useful for sharing files by ensuring that all access to files on
  # service will use the named group for their permissions checking.
  #   Thus, by assigning permissions for this group to the files and
  #   directories within this service the Samba administrator can
  #   restrict or allow sharing of these files.
  force group = dycom-all-users

##UPDATED FOR 2NODE HA CLUSTER
# backup space for Heidi's laptop (windows macrium images & diffs)
[backups_heidi]
  comment = Backups Directory to Store Macrium Images files
  path = /mnt/drbd/3TB/storage/backups_heidi
  read only = no
  create mask = 0770
  force create mode = 0770
  directory mask = 0770
  force directory mode = 0770
  #*** uncomment the command below if you struggle with permissions (see
  #man smb.conf) ***
  ;force group = dycom
```

```
##UPDATED FOR 2NODE HA CLUSTER
# Backup space for workstation no.1 (labeled 8113)
# Note: directory not broadcast (visible) on network, but accessible to
# all users on local share.
[backups_ws1_8113]
  comment = Backups Directory (hidden) - only for workstation HP.i7.8113
  path = /mnt/drbd/3TB/storage/backups_ws1_8113
  read only = no
  browsable = no
  create mask = 0770
  force create mode = 0770
  directory mask = 0770
  force directory mode = 0770
  force group = dycom-and-ws1

##UPDATED FOR 2NODE HA CLUSTER
# Backup space for workstation no.2 (s/n AUD3380RXR)
# Note: directory not broadcast (visible) on network, but accessible to
# all users on local share.
[backups_ws2_aud3380]
  comment = Backups Directory (hidden) - for workstation AUD3380RXR
  path = /mnt/drbd/3TB/storage/backups_ws2_aud3380
  read only = no
  browsable = no
  create mask = 0770
  force create mode = 0770
  directory mask = 0770
  force directory mode = 0770
  force group = dycom-and-ws2

##UPDATED FOR 2NODE HA CLUSTER
# Backup space for workstation no.2 (s/n EXWKSHP8203)
# Note: directory not broadcast (visible) on network, but accessible to
# all users on local share.
[backups_ws3_8203]
  comment = Backups Directory (hidden) - for workstation EXWKSHP8203
  path = /mnt/drbd/3TB/storage/backups_ws3_8203
  read only = no
  browsable = no
  create mask = 0770
  force create mode = 0770
  directory mask = 0770
  force directory mode = 0770
  force group = dycom-and-ws3

##UPDATED FOR 2NODE HA CLUSTER
##READ-ONLY share of Incremental "snapshot" backups directory of
##Dycom's 'shared_bkp' folder
[shared-backups]
  comment = Incremental "snapshot" Backups Directory (read only)
  path = /mnt/drbd/3TB/storage/shared_bkp

  #catia vfs module - to map Windows-incompatible special characters to
  #compatible alternatives (e.g. ':' to '÷')
  vfs objects = catia
  catia:mappings =
0x22:0xa8,0x2a:0xa4,0x2f:0xf8,0x3a:0xf7,0x3c:0xab,0x3e:0xbb,0x3f:0xbf,0x5c:0xff
,0x7c:0xa6

  browsable = yes
  read only = yes
  ;writable = no
  read list = @dycom
  #uncomment below if you have problems with access
  valid users = @dycom

  #alternative values & options (more than one group (preceded with '@')
  #or user simultaneously)
  ;read list = @dycom @dycom-all-users
  ;valid users = @dycom @dycom-all-users

##UPDATED FOR 2NODE HA CLUSTER
##Backup space for student workstations (windows macrium images & diffs)
[backups_student_workstations]
  comment = Backups Directory to Store Macrium Images for student workstations
  path = /mnt/drbd/3TB/storage/backups_student_workstations
  read only = no
  create mask = 0770
  force create mode = 0770
  directory mask = 0770
  force directory mode = 0770
  #*** uncomment the command below if you struggle with permissions (see
  #man smb.conf) ***
  force group = dycom
  valid users = heidi andre
```

Important to note that the `path` parameter points to the file path to where the DRBD device (effectively the backing block device) has been mounted to the file system. Keep in mind that the DRBD-device acts as an interface to its backing block device, where the actual data is stored. Thus, transactions (read/write) to the DRBD-device are actually performed on the backing block device.

Install Pacemaker & Corosync

On both nodes we must first ensure that the DRBD service is not running (nor enabled on startup) on either node.

- `sudo systemctl disable drbd`
  OR
- `sudo systemctl stop drbd`

We should also ensure that the DRBD device is neither mounted nor enabled on either node:

- `sudo umount<drbd-device-mountpoint>`
  - o eg.:   `sudoumount /mnt/drbd/1G-USB`
- `sudo drbdadm down <resource>`
  - o eg.:   `sudodrbdadm down r0`

Then we can install Pacemaker and Corosyncon both nodes:

- `sudo apt-get install pacemaker corosync -y`

After the installation, both these services are running automatically on the system. Stop them with the `systemctl` commands below:

- `sudo systemctl stop corosync`
- `sudo systemctl stop pacemaker`

NOTE: *systemctl* also enables/disables a service as part of the system startup

Confirm both services are stopped with:

- `sudo systemctl status corosync`
- `sudo systemctl status pacemaker`

Configure Corosync

First backup the `/etc/corosync/corosync.conf` file on both devices

Open the configuration file for edit:

- `sudo nano /etc/corosync/corosync.conf`

Now add or update the configurations below into that file:

Note that for simplicity we are using a non-encryption setup and assume isolated LAN connections (ie. direct NIC to NIC communication). For secure authenticated communications – such required when nodes are on a WAN or remote location - we recommend reviewing literature on shared-key encryption with the "corosync-keygen" utility. The relevant section in "How to Set up Nginx High Availability with Pacemaker, Corosync, and Crmsh on Ubuntu 16.04" tutorial found elsewhere, or another from this link give a good starting point on how to accomplish this.

```
# Totem Protocol Configuration – Totem Protocol is the native protocol
# used by Corosync.  lease read the corosync.conf.5 manual page
totem {
version: 2

  # Corosync itself works without a cluster name, but DLM needs one.
  # The cluster name is also written into the VG metadata of newly
  # created shared LVM volume groups, if lvmlockd uses DLM locking.
  # It is also used for computing mcastaddr, unless overridden below.
  cluster_name: dycom-cluster

  # crypto_cipher and crypto_hash: Used for mutual node authentication.
  # If you choose to enable this, then do remember to create a shared
  # secret with "corosync-keygen".
  # enabling crypto_cipher, requires also enabling of crypto_hash.
  # crypto_cipher and crypto_hash should be used instead of deprecated
  # secauth parameter.
  secauth: off
  transport: udpu

  # Valid values for crypto_cipher are none (no encryption), aes256,
  # aes192, aes128 and  3des. Enabling crypto_cipher, requires also
  # enabling of crypto_hash.
  crypto_cipher: none

  # Valid values for crypto_hashare  none  (no  authentication),  md5,
  # sha1, sha256, sha384 and sha512.
  crypto_hash: none

  # Network Interface configuration for Corosync.
  # Define at least one interface to communicate over.
  # If you define more than one interface stanza, you must
  # also set rrp_mode.
  interface {
     # Rings must be consecutively numbered, starting at 0.
```

```
          ringnumber: 0

          # This is normally the *network* address of the interface to bind to.
          # This ensures that you can use identical instances of this
          # configuration file across all your cluster nodes, without having
          # to modify this option.
          bindnetaddr: 10.1.1.0
          #bindnetaddr: 192.168.178.0

          # However, if you have multiple physical network interfaces
          # configured for the same subnet, then the network address alone is
          # not sufficient to identify the interface Corosync should bind to.
          # In that case, configure the *host* address of the interface
          # instead:
          # bindnetaddr: 10.1.1.1
          # bindnetaddr: 192.168.178.30

          broadcast: yes

          # Corosync uses the port you specify here for UDP messaging, and
          # also the immediately preceding port. Thus if you set this to 5405,
          # Corosync sends messages over UDP ports 5405 and 5404.
          mcastport: 5405
     }
     #e/o interface section
}
#e/o totem object

# Nodelist - ie. Servers in the Cluster
nodelist {
          node {
               #the IP address of the network interface used
               ring0_addr: 10.1.1.1
               #ring0_addr: 192.168.178.30

               #optional: the host name of the node at the above address
               name: dycom-master

               #optionally assign a fixed note id (integer) - comment out
               #when troubleshooting!!!!
               #nodeid: 0
     }
     node {
               ring0_addr: 10.1.1.2
               #ring0_addr: 192.168.178.54
               name: dycom-slave
               #nodeid: 1
     }
     #more nodes can be added here….
}

# Quorum configuration
quorum {
          # Enable and configure quorum subsystem (default: off)
          # see also corosync.conf.5 and votequorum.5
          provider: corosync_votequorum
          #expected_votes: 2

          #must be defined for 2-node clusters (and ONLY for 2-node clusters!!)
          two_node:  1
```

```
        #OPTIONAL:
        wait_for_all: 1
        last_man_standing: 1
        auto_tie_breaker: 0
}

# OPTIONAL:  Corosync Log configuration - potentially usefull for debugging
logging {
        # Log the source file and line where messages are being generated.
        # When in doubt, leave off. Potentially useful for debugging.
        fileline: off

        # Log to standard error. When in doubt, set to no. Useful when
        # running in the foreground (when invoking "corosync -f")
        to_stderr: no

        # Log to a log file. When set to "no", the "logfile" option must not
# be set.
        to_logfile: yes
        logfile: /var/log/corosync/corosync.log

        # Log to the system log daemon. When in doubt, set to yes.
        to_syslog: yes

        # Log messages with time stamps. When in doubt, set to on
        # (unless you are only logging to syslog, where double
        # timestamps can be annoying).
        timestamp: on
        logger_subsys {
                subsys: QUORUM
                debug: off
        }
}
#e/o logging object

##OPTIONAL
#service {
#       name: pacemaker
#       ver: 0
#}
```

In the above configuration it is vital to use the network interface IP address for the nodes in question, and that you use the network address for the *bindnetaddr* variable. The latter is usually the same as those used for network interfaces on the nodes, but with a trailing 0.

Eg.:   10.1.1.0
        192.168.178.0

Set Firewall Rules

On both nodes, we are going to add the firewall rules to allow through traffic between the nodes. We do however not open any port on the router or gateway to the internet, because we have no security in place to protect the port from public access.

Corosync uses UDP by default. It uses both the port specified in the `corosync.conf` file, and the port directly before that. Eg. if you specified port 5405, then it will use port 5404 as well. Both these ports will have to be opened/forwarded on the firewall:
Eg.:

- `sudo ufw allow 5404/udp`
- `sudo ufw allow 5405/udp`

If not already configured earlier; let's allow all TCP traffic through firewall at ports 139 and 445 to the Samba server (natively required for Samba to work)

- `sudo ufw allow 139/tcp`
- `sudo ufw allow 445/tcp`

And finally, also forward ports for SSH, so that we can gain access from a remote location. NOTE that with SSH we assume a secure key authentication method is being used as explained in another of our tutorials called *Ubuntu Server 16.04 LTS Installation & Configuration*. Also note that the port will have to be forwarded on the gateway router firewall for access from the internet as well.
Because we'd like to have access to both primary and secondary nodes at any stage; we'll have to open separate ports for SSH on each node,

eg.:

- `sudo ufw allow 2202/tcp          #on node 1`
- `sudo ufw allow 2203/tcp          #on node 2`

Start Pacemaker & Corosync

Up until notified you are to assume that the DRBD is not running, so go ahead and stop the service if it isn't already.

We now start/restart Corosync on both nodes:

- `sudo systemctl [re]start corosync`

Followed by starting Pacemaker on both nodes:

- `sudo systemctl start pacemaker`

The order in which Pacemaker and Corosync is started or stopped is based on stacking of these services. It would appear that it doesn't matter, but with Corosync being the messaging layer it would make sense to start Corosync first, and stop it last.

Configure Pacemaker (CRM)

On either host we can use the command *crm status* to see the cluster come online.

- `sudo crm status`

In the output of the above command, you should notice that there are no cluster resources listed at this time. It should also show the number of nodes (eg. 2) being `Online`. If this is not the case – STOP – review, troubleshoot, repeat and continue only when this is the case.

The primary goal of any CRM is to define and manage cluster resources. On the **primary node ONLY** - we can now set some of the cluster properties and create cluster resources.

Command Syntax Reference

The syntax used to describe any command henceforth will include some simple tokens:

- <> denotes a required string, eg.: name or id,
- `[]` means that the construct is optional,
- the ellipsis or parantheses `(...)` signifies that the previous construct may be repeated,
- `|` means pick one of many,
- the double colon `::` signifies an attempt to elaborate on a command, property or attribute. It is not part of an actual command, and is purely included for clarification.
  - **Note to self:** this seems to be the case when looking at the examples in the underlying source article used for this manual. Has to be confirmed still.
- and the rest are literals (*strings*, `:`, =).

Sample CIB Configuration

Using the requirements described at the start of this project – a two-node cluster using DRBD on local ext4 formatted backing block device, Pacemaker/Corosync, and Samba fileserver in active/passive redundancy configuration – we can now configure the Pacemaker CRM using the *crmsh* interactive shell. To see the full breakdown explained in detail, please consult the tutorial/manual:

- `sudo crm configure`

```
crm(live)# property stonith-enabled=false
crm(live)# property no-quorum-policy=ignore
crm(live)# property cluster-name=dycom-cluster

crm(live)# primitive drbd_res ocf:linbit:drbd  \
      params  drbd_resource=r0 \
      op monitor  interval=29s role=Master  \
      op  monitor  interval=31s  role=Slave

crm(live)# ms  drbd_master_slave  drbd_res \
      meta  master-max=1  master-node-max=1 \
            clone-max=2  clone-node-max=1  notify=true

crm(live)# primitive  fs_res ocf:heartbeat:Filesystem \
      params  device=/dev/drbd0 \
              directory=/mnt/drbd/3TB \
              fstype=ext4

crm(live)# primitive samba_res lsb: smbd
      meta migration-threshold="10"

crm(live)# collocation  fs_drbd_colo  INFINITY: fs_res  drbd_master_slave:Master
crm(live)# order  fs_after_drbd  mandatory: drbd_master_slave:promote  \
      fs_res:start

crm(live)# collocation  samba_drbd_colo  INFINITY: \
      samba_res drbd_master_slave:Master

crm(live)# order  samba_after_drbd  mandatory: \
      drbd_master_slave:promote samba_res:start

crm(live)# primitive virtual_ip ocf:heartbeat:IPaddr2 \
      params ip="192.168.178.201" \
            cidr_netmask="24"
      op monitor interval="10s" \
      meta migration-threshold="10"

crm(live)# group virtual_ip_samba virtual_ip samba_res

crm(live)# primitive sysadmin_MailTo ocf:heartbeat:MailTo \
      params email=skyqode@gmail.com\
      op monitor depth="0" timeout="10" interval="10"

crm(live)# collocation  mailto_drbd_colo  INFINITY: \
      sysadmin_MailTo drbd_master_slave:Master

crm(live)# order mailto_after_drbd  mandatory: \
      drbd_master_slave:promote sysadmin_MailTo:start


crm(live)# commit
crm(live)# show
crm(live)# quit
```

NOTE that all of the properties entered using the interactive *CRM Shell*, can also be added (or amended) by invoking the *crm configure* command on a one-by-one case, directly at the native shell prompt. If done in this way, the `commit` command is not required because every command completed will automatically invoke a `commit`:

eg:

```
sudo crm configure property stonith-enabled=false
sudo crm configure property no-quorum-policy=ignore
sudo crm configure property cluster-name=dycom-cluster

sudo crm configure primitive drbd_res ocf:linbit:drbd params drbd_resource=r0 op monitor interval=29s role=Master op monitor interval=31s role=Slave
sudo crm configure ms drbd_master_slave drbd_res meta master-max=1 master-node-max=1 clone-max=2 clone-node-max=1 notify=true
sudo crm configure primitive fs_res ocf:heartbeat:Filesystem params device=/dev/drbd0 directory=/mnt/drbd/1G-USB fstype=ext4
sudo crm configure primitive samba_res lsb: smbd meta migration-threshold="10"
sudo crm configure collocation fs_drbd_colo INFINITY: fs_res drbd_master_slave:Master
sudo crm configure order fs_after_drbd mandatory: drbd_master_slave:promote fs_res:start
sudo crm configure collocation  samba_drbd_colo  INFINITY: samba_res drbd_master_slave:Master
sudo crm configure order  samba_after_drbd  mandatory: drbd_master_slave:promote samba_res:start

sudo crm configure primitive virtual_ip ocf:heartbeat:IPaddr2 params ip="192.168.178.201"  cidr_netmask="24"  op monitor interval="10s" meta migration-
threshold="10"
sudo crm configure group virtual_ip_samba virtual_ip samba_res

sudo crm configure primitive sysadmin_MailTo ocf:heartbeat:MailTo params email=skyqode@gmail.com op monitor depth="0" timeout="10" interval="10"
sudo crm configure collocation  mailto_drbd_colo  INFINITY: sysadmin_MailTo drbd_master_slave:Master
sudo crm configure order  mailto_after_drbd  mandatory: drbd_master_slave:promote sysadmin_MailTo:start
sudo crm configure commit
sudo crm configure show
```

CRM Monitoring & Navigation

If you have reached this stage and had correctly configured your CRM, you should have a CRM that will manage all cluster resources and underlying actions. I repeat: **No further management of cluster resources should be administered either at system start up or manually; the CRM assumes full control of these resource.** Only defer from this when troubleshooting errors. In such a case, follow the strategies outlined in the *Troubleshooting* section below.

show

To view the current configuration committed to the CIB, execute:

- `sudo crm configure show`

Executing this command on either primary or secondary node(s) should reflect all the properties entered during the configuration phase on the primary node earlier. Recall that the configuration is relayed to all secondary nodes automatically.

crm_mon

You can monitor the cluster with a summary output using either CRM commands `crm status` or `crm_mon`:

- `sudo crm_mon`

For cluster monitoring - incl. that of DRBD, Pacemaker (CRM), Corosync and Samba - I prefer to have a *tmux* window open with all of the following panes to give a broad spectrum overview of the entire cluster and its resources:

> LHS1 `watch -n 10.0 'sudo systemctl status drbd'`

- to confirm DRBD service (script drbd) has been loaded at start up, and is active

> LHS2 `watch -n 10.0 'sudo cat /proc/drbd'`

- use pseudo file system `/proc` to view DRBD status (connected, roles, replication status)

> LHS3 `watch -n 10.0 'ls -alth <drbd-device-mountpath>'`

- eg. `'ls -alth /mnt/drbd/1G-USB'` to list contents of DRBD backing block device (ie. the storage space that'll be replicated to secondary node(s))

> LHS4 `watch -n 10.0 'sudo crm configure show'`

- to view the current CRM configuration

> LHS5 `watch -n 10.0 systemctl status smbd`

- to view Samba status

> RHS1 `watch -n 10.0 'sudo systemctl status pacemaker'`

- to view the running status of the Pacemaker CRM (Cluster Resource Manager) service

> RHS2 `watch -n 10.0 'sudo systemctl status corosync'`

- to view messaging layer for CRM

> RHS3 `sudo crm_mon`

- CRM monitor - ie monitor status of cluster nodes, node roles, file system status - alt.: watch -n 5.0 'sudo crm status'

> RHS4 `watch -n 5.0 'sudo cat /var/log/corosync/corosync.log | grep -Ei "(crit|warning|error)"'`

- monitor corosync log file for any critical errors and warnings

> RHS5 leave this pane open for CRM or DRBD commands

eg.:

```
sudo systemctl stop corosync && sudo systemctl stop pacemaker
```

- stop CRM locally to effectively leave cluster (relinquish primary role)

```
sudo systemctl start corosync && sudo systemctl start pacemaker
```

- start CRM locally to effectively rejoin cluster (in secondary role)

```
sudo drbdadm [up|down] r0
```

- *bring up* or *bring down* the DRBD resource `r0`
  - resource label found in */etc/drbd.conf*

```
sudo systemctl [stop|start] drbd
```

- stop/start DRBD locally to effectively enable/disable drbd at startup
- this might also “bring up” or “bring down” the DRBD resource

CRM Testing

To see a set of troubleshooting methods, please review the appropriate section in the tutorial/manual.

Prelim

If we were to run `ls` command on the mounted FS on both primary and secondary node; we'll see that only the primary node (as confirmed in the `crm_mon` command output) will show the content of that mounting; the secondary node never has the resource mounted to the FS.

- `ls -alth <fs-drbd-device-mount-point>`
  - eg.: `ls -alth /mnt/drbd/3TB`

Fail-Over

Test 1

So let's now test that the CRM is working, by wilfully rebooting the primary node, and then observe both the CRM and DRBD status on the secondary node:

- `sudo shutdown -r now`
  OR
- `sudo reboot`

Running `crm_mon` now on the secondary node – you should see that it has been promoted to the master role, and the former primary/master node is now both OFFLINE and demoted to the slave/secondary role (or eventually you'll see this after it's back online).

- `sudo crm_mon`

Test 2

A similar scenario – without any reboot - could be tested by simply stopping both `pacemaker` and `Corosync` services on the primary node, followed by (if you want) starting both services up again.

In either test-case 1 or 2; failure or stoppage of Corosync on the primary node should trigger Corosync on the secondary node to relay the failure to the CRM (Pacemaker) on the secondary node. The CRM on the secondary node should then start the fail-over processes in the manner set out in the CIB configuration. Both the CRM and DRBD status should reflect a successful connected state and role relationship between the nodes. Eg.:

- `sudo cat /proc/drbd`

  example output:

```
version: 8.4.5 (api:1/proto:86-101)
srcversion: 611D9EEFB9C11D2BC709D07
 0: cs:WFConnection ro:Primary/Unknown ds:UpToDate/DUnknown C r-----
    ns:0 nr:8 dw:12 dr:673 al:1 bm:0 lo:0 pe:0 ua:0 ap:0 ep:1 wo:f oos:4
```

- `sudo crm_mon`

example output:

```
Stack: corosync
Current DC: dycom-slave (version 1.1.14-70404b0) - partition with quorum
2 nodes and 6 resources configured

Online: [ dycom-slave ]
OFFLINE: [ dycom-master ]

 Master/Slave Set: drbd_master_slave [drbd_res]
     Masters: [ dycom-slave ]
fs_res  (ocf::heartbeat:Filesystem):    Started dycom-slave
 Resource Group: virtual_ip_samba
     virtual_ip (ocf::heartbeat:IPaddr2):       Started dycom-slave
     samba_res  (lsb:smbd):     Started dycom-slave
```

Incremental Backups and Hard Links in the Cluster

An incremental backup – one that utilizes *hardlinks* to retain the file structure of the original source - is implemented to save *snapshot* backups of the `/mnt/drbd/3TB/storage/shared` directory. The backup script is called as a *cronjob* every 2 hours, and all snapshots are saved to `/mnt/drbd/3TB/backups/full-and-incremental/shared-bkp`. Every backup will be stored under unique month-day-year-time named directories, and will preserve the original directory structure. When traversing any of the backup directories, you'd see every file from the source directory exactly as it was at that time. Yet, there would be no duplicates across any two directories. RSYNC accomplishes this with the use of hardlinking through the `--link-dest=<directory>` command argument.

Backup Script Considerations

Hard Links

Using *hardlinks* in a file system on a clustered node has no bearing on the DRBD replication process, as long as DRBD and the backing block device is configured identically on both nodes - as is described earlier.

Backup Script Locations

Backup scripts - even incremental backup scripts which utilize RSYNC, date-stamped, hard -link based code - can be used in clustered nodes, **but only if** the backup script gets executed on the node that has the DRBD device mounted to the file system. All sorts of data corruption risks - including complete data deletion - may be at risk should you not heed this warning.

In the tutorial/manual there are mentioned three cluster implementations for incremental backup strategies where only the primary node has the ability to run the backup script. In this project, the 1st method is employed (see next). The script itself is constantly being tweaked for improvements, and due to the sheer size of it cannot be included in this document.

Method 1: backup script located on DRBD backing block device mounted on file system

- Note that the script employed depends on specific supporting files and directories. It would be a good idea to have these also placed within the file system (of the backing block device) that is mounted to the file system at */mnt/drbd/3TB* – even if just for increased security
  - Note to self:
    - In the current version of your incremental backup script, this would include directories `./backup-info` and `./files` along with its nested files and scripts

- Considered the easiest solution and quite safe – it **doesn't require any changes to the backup script**. Both nodes in the cluster have exactly the same script, and is scheduled (eg. as cronjob) to run exactly the same times on both nodes.
  - When the script is scheduled to run on the *primary* node, it is executed from where it is located on the mounted file system. No issues – the script runs as it should – and DRBD will replicate in real time to the secondary node.
  - When the script is scheduled to run on the *secondary* node, the script won't be found due to the file system not being mounted, and the session that was supposed to run the script will fail silently. No issues again - the script never executes, because the script isn't available.
  - This way the backup will run – as it should – only on the node that has the file system mounted!!!