

UNIVERSITÀ DEGLI STUDI DI PERUGIA  
CALCOLO DISTRIBUITO E SISTEMI AD ALTE PRESTAZIONI,  
PRIMO SEMESTRE 2018

---

## Configuration of a High Availability Cluster

---



Yana Kozak  
October 21, 2018

# 1 Purpose

High availability systems play a very important role nowadays as there do exist system components, such as critical databases, network level file sharing, business applications or customer services (electronic commerce websites for example), that result to be essential to business operation and should be up 24h a day - downtimes are unthinkable in these cases. HA systems infrastructure that assures the continuous functioning of these components.

This project's objective is the configuration of a 2-node cluster environment providing a continued service - that is with a minimum amount of down time in case of failure. Two Fedora Server 28 has been configured on Oracle VM VirtualBox (running on Windows 10) in order to implement this type of fail-over cluster. We will use 3 specific software applications to accomplish this task: Pacemaker, Corosync and DRBD which are explained immediately in this section.

## 1.1 Pacemaker

Pacemaker<sup>1</sup> is an open source software for managing resources of a high-availability cluster: it is run on a set of hosts - or nodes - making part of the cluster in such a way that the downtime of the desired services is minimized. It has been released in 2004 as part of Linux-HA project<sup>2</sup> - that aimed at providing a high availability, scalable solution to Linux and many Unix variants. What does it do? It provides different cluster management tools and some of its functionalities are listed below

- Performs detection and recovery from hardware and software failures.
- Ensures data integrity by isolating the undermined hosts - for this purpose **STONITH** (an acronym for Shoot The Other Node In The Head) is used (to ensure that it is not possible for a node to be running a service<sup>3</sup>).
- Allows shared storage.
- Automatic replication of the configurations that have been updated from any of the nodes of the cluster.

It is usually used with Corosync Cluster Engine or Linux-HA Heartbeat.

## 1.2 Corosync

Corosync is an open source group communication system which counts among its additional features high availability implementation. It has been released in 2008 and derives from openAIS project<sup>4</sup> - which was founded in 2002 to implement Service Availability Forum Application Interface Specification (SaForumAIS) - SaForum is an organization that develops specifications for carrier-grade and critical-mission systems. It neatly separates what constitutes core infrastructure from the very cluster services

---

<sup>1</sup><https://github.com/ClusterLabs/pacemaker>

<sup>2</sup><http://www.linux-ha.org/>

<sup>3</sup><http://clusterlabs.org/>

<sup>4</sup><http://www.openais.eu/en/project-overview/>

## 1.3 DRBD

Distributed Replicated Block Device<sup>5</sup> is a free-software distributed storage system for Linux which allows either of asynchronous and synchronous block storage replication between servers. It contains numerous applications for userspace management and is mainly used in High Availability clusters. Its first release dates back to 2007 and now it is developed by LINBIT<sup>6</sup> - a software company specialized in data replication - among which block storage.

## 2 Installation of Fedora Server OS on a VirtualBox machine

Resources that have been allocated for each virtual machine<sup>7</sup>:

- RAM: 4 GB
- Hard Disk: 20 GB

The primary server corresponds to `nodol` virtual machine and it has been cloned to create a secondary one, hosted inside `node2` VM - see Figure 1. Both VMs have been assigned a static hostname - `sanjunipero` and `westworld` respectively - which have been reused also for name resolution in static IP assignment, which we will see later:

```
hostnamectl set-hostname sanjunipero
hostnamectl set-hostname westworld
```

## 3 Network configuration

Two network interface cards for both of the virtual machines have been created: for the 1st NIC `Host-only networking` mode has been used, while for the 2nd NIC we resorted to `NAT` mode.

### 3.1 Host-only networking

When host-only networking is used, VirtualBox creates a new software interface on the host which then appears next to your existing network interfaces. Host-only networking is particularly useful for preconfigured virtual appliances, where multiple virtual machines are shipped together and designed to cooperate.

Before you can attach a VM to a host-only network you have to `create a host-only interface` - see Figures 3 and 4. For host-only networking, the `DHCP server that is built into VirtualBox` - see the checked box in Figure 4 - has been used. It has been enabled to manage the IP addresses in the host-only network since otherwise we would need to configure all IP addresses statically.

---

<sup>5</sup><https://github.com/LINBIT/drbd-9.0>

<sup>6</sup><https://www.linbit.com/en/>

<sup>7</sup><https://www.oracle.com/technetwork/server-storage/virtualbox/downloads/index.html>

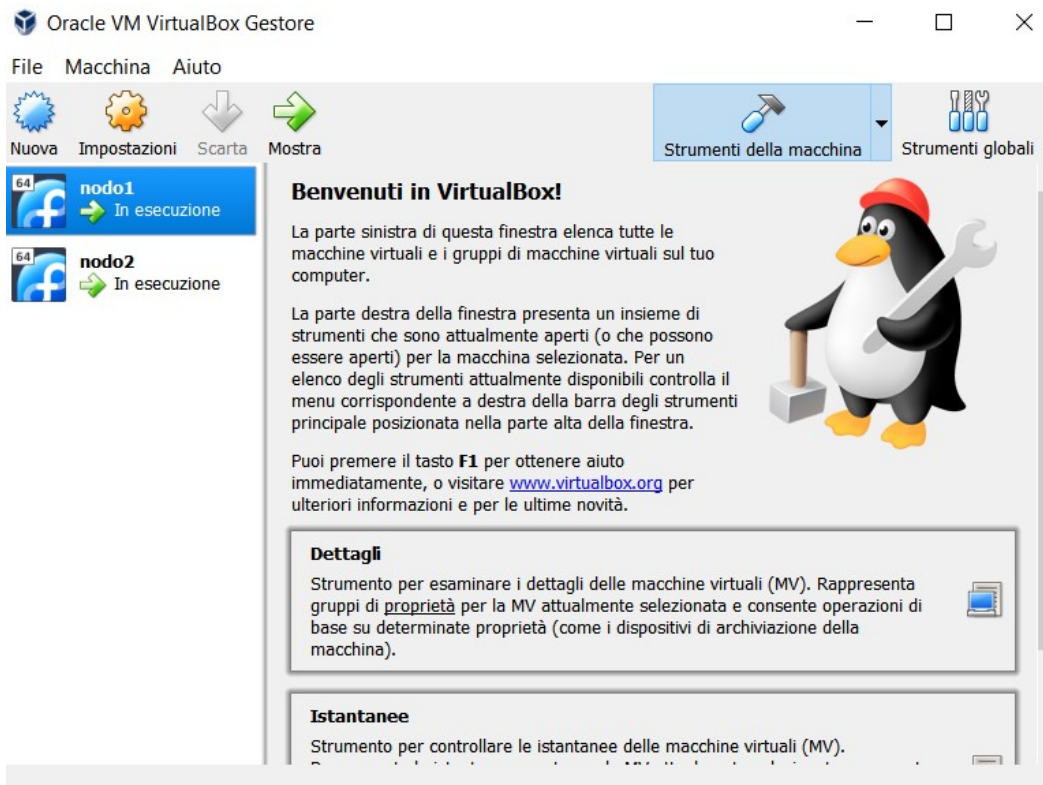


Figure 1: Two virtual machines with Fedora Server OS installed

### 3.2 NAT networking

Network Address Translation (NAT) is the simplest way of [accessing an external network from a virtual machine](#). A virtual machine with NAT enabled acts much like a real computer that connects to the Internet through a router. The “router”, in this case, is the VirtualBox networking engine, which maps traffic from and to the virtual machine transparently. In VirtualBox this router is placed between each virtual machine and the host. This separation maximizes security since by default virtual machines cannot talk to each other. The disadvantage of NAT mode is that, much like a private network behind a router, the virtual machine is invisible and unreachable from the outside internet; you cannot run a server this way unless you set up port forwarding. The network frames sent out by the guest operating system are received by VirtualBox’s NAT engine, which extracts the TCP/IP data and resends it using the host operating system. To an application on the host, or to another computer on the same network as the host, it looks like the data was sent by the VirtualBox application on the host, using an IP address belonging to the host. VirtualBox listens for replies to the packages sent, and repacks and resends them to the guest machine on its private network. The virtual machine receives its network address and configuration on the private network from a DHCP server integrated into VirtualBox. The IP address thus assigned to the virtual machine is usually on a completely different network than the

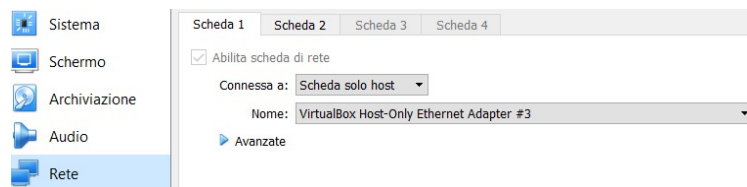


Figure 2: Creation of the Host-only network card for the guest machine.

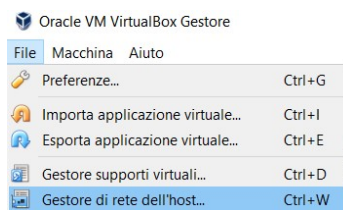


Figure 3: Accessing the network configuration of the host.

host.



Figure 4: Creation of the Host-only network with the pre-built DHCP server enabled.



Figure 5: Creation of a NAT NIC for the guest.

Port	When Required
TCP 2224	<p>Required on all nodes (needed by the <code>pcsd</code> Web UI and required for node-to-node communication)</p> <p>It is crucial to open port 2224 in such a way that <code>pcs</code> from any node can talk to all nodes in the cluster, including itself. When using the Booth cluster ticket manager or a quorum device you must open port 2224 on all related hosts, such as Booth arbiters or the quorum device host.</p>
TCP 3121	<p>Required on all nodes if the cluster has any Pacemaker Remote nodes</p> <p>Pacemaker's <code>crmd</code> daemon on the full cluster nodes will contact the <code>pacemaker_remoted</code> daemon on Pacemaker Remote nodes at port 3121. If a separate interface is used for cluster communication, the port only needs to be open on that interface. At a minimum, the port should open on Pacemaker Remote nodes to full cluster nodes. Because users may convert a host between a full node and a remote node, or run a remote node inside a container using the host's network, it can be useful to open the port to all nodes. It is not necessary to open the port to any hosts other than nodes.</p>
TCP 5403	<p>Required on the quorum device host when using a quorum device with <code>corosync-qnetd</code>. The default value can be changed with the <code>-p</code> option of the <code>corosync-qnetd</code> command.</p>
UDP 5404	<p>Required on corosync nodes if <code>corosync</code> is configured for multicast UDP</p>
UDP 5405	<p>Required on all corosync nodes (needed by <code>corosync</code>)</p>
TCP 21064	<p>Required on all nodes if the cluster contains any resources requiring DLM (such as <code>clvm</code> or <code>GFS2</code>)</p>

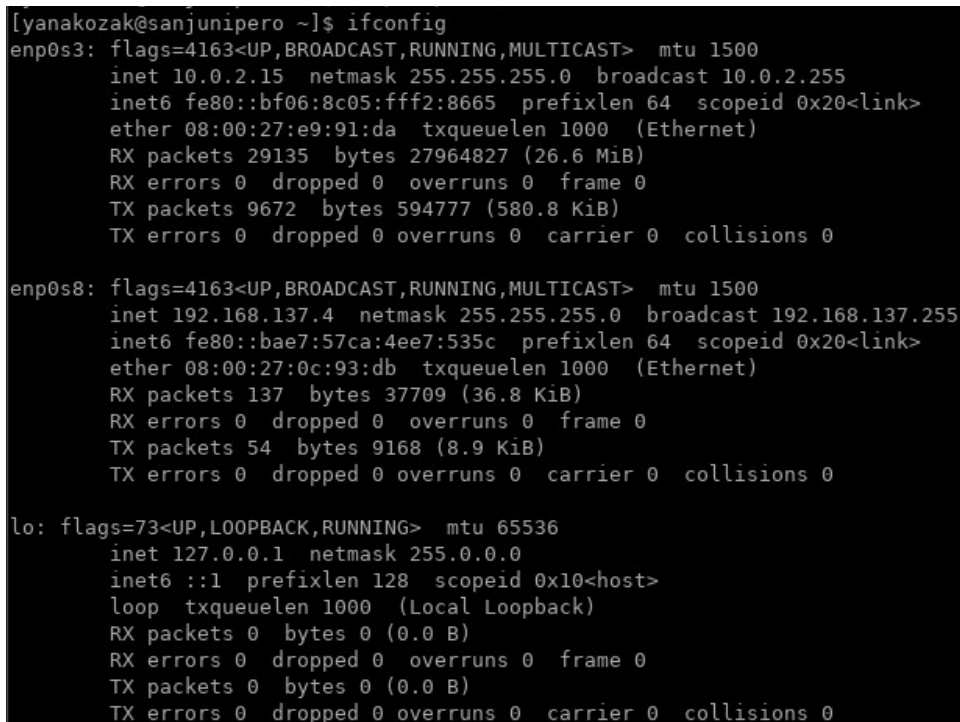
Figure 6: Ports that need to be enabled for an HA cluster



**Configuring port forwarding with NAT.** As the virtual machine is connected to a private network internal to VirtualBox and invisible to the host, network services on the guest are not accessible to the host machine or to other computers on the same network. However, like a physical router, VirtualBox can make selected services available to the world outside the guest through port forwarding. This means that VirtualBox listens to certain ports on the host and resends all packets which arrive there to the guest, on the same or a different port. Port forwarding on uor VMs has been resolved with these commands:

```
firewall-cmd --permanent --add-service=high-availability
firewall-cmd --add-service=high-availability
```

In Figure 6 all the ports that have been enabled by the previous two commands are described<sup>8</sup>.

A terminal window showing the output of the 'ifconfig' command on a system named 'sanjunipero'. The output displays details for three network interfaces: 'enp0s3', 'enp0s8', and 'lo'. Each interface shows its flags, MTU, IP address, netmask, broadcast address, MAC address, and statistics for RX and TX packets, bytes, errors, and drops. 'enp0s3' is an Ethernet interface with IP 10.0.2.15. 'enp0s8' is an Ethernet interface with IP 192.168.137.4. 'lo' is a loopback interface with IP 127.0.0.1.

```
[yanakozak@sanjunipero ~]$ ifconfig
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
    inet 10.0.2.15  netmask 255.255.255.0  broadcast 10.0.2.255
    inet6 fe80::bf06:8c05:fff2:8665  prefixlen 64  scopeid 0x20<link>
    ether 08:00:27:e9:91:da  txqueuelen 1000  (Ethernet)
    RX packets 29135  bytes 27964827 (26.6 MiB)
    RX errors 0  dropped 0  overruns 0  frame 0
    TX packets 9672  bytes 594777 (580.8 KiB)
    TX errors 0  dropped 0  overruns 0  carrier 0  collisions 0

enp0s8: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
    inet 192.168.137.4  netmask 255.255.255.0  broadcast 192.168.137.255
    inet6 fe80::bae7:57ca:4ee7:535c  prefixlen 64  scopeid 0x20<link>
    ether 08:00:27:0c:93:db  txqueuelen 1000  (Ethernet)
    RX packets 137  bytes 37709 (36.8 KiB)
    RX errors 0  dropped 0  overruns 0  frame 0
    TX packets 54  bytes 9168 (8.9 KiB)
    TX errors 0  dropped 0  overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
    inet 127.0.0.1  netmask 255.0.0.0
    inet6 ::1  prefixlen 128  scopeid 0x10<host>
    loop txqueuelen 1000  (Local Loopback)
    RX packets 0  bytes 0 (0.0 B)
    RX errors 0  dropped 0  overruns 0  frame 0
    TX packets 0  bytes 0 (0.0 B)
    TX errors 0  dropped 0  overruns 0  carrier 0  collisions 0
```

Figure 7: Details about the configured Network Interface Cards of *sanjunipero*.

### 3.3 Static IP addresses and Name Resolution

As we have only two nodes in our network, it is not necessary to use any routing protocol - we assign manually the IP addresses and the relative hostnames by which the machines with those IPs should be identified - see Figure 8 - so that we do not have to write numbers

<sup>8</sup>This image was taken from [https://access.redhat.com/documentation/en-us/red\\_hat\\_enterprise\\_linux/7/html/high\\_availability\\_add-on\\_reference/s1-firewalls-haar](https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/high_availability_add-on_reference/s1-firewalls-haar)

each time we need to connect them to. Figure 7 shows how our network cards have been configured.

```
127.0.0.1    localhost localhost.localdomain localhost4 localhost4.localdomain
::1        localhost localhost.localdomain localhost6 localhost6.localdomain
192.168.137.3 westworld
192.168.137.4 sanjunipero
```

Figure 8: Modification of `/etc/hosts` file of *sanjunipero*.

## 4 Installing packages for HA cluster

We will need the following libraries:

- `sudo dnf -y install pacemaker corosync pcs`
- `sudo dnf -y install drbd-pacemaker drbd-udev` (shared file-system between machines of the clusters)
- `sudo dnf -y install httpd`
- `sudo dnf -y install iptables-services`

It should be noted that on Fedora Server the last two packages come preinstalled already!

## 5 Creation of the cluster

When we installed the packages mentioned at the beginning of Section 4, `dnf` created automatically a user called `hacluster` which can be utilized together with `pcs` to manage cluster nodes.

1. Let's give our user a password on both nodes so that we can authenticate:

```
[yanakozak@sanjunipero ~]$ sudo passwd hacluster
[sudo] password di yanakozak:
Cambio password per l'utente hacluster.
Nuova password:
Reimmettere la nuova password:
passwd: tutti token di autenticazione sono stati aggiornati con successo.
```

2. Then start `pcsd` service on both nodes with the following commands:

```
sudo systemctl start pcsd.service
sudo systemctl enable pcsd.service
```

3. Now, as we want to manage our cluster from one single point, we need to authenticate on all of the nodes (we have authenticated previously):



```
[yanakozak@sanjunipero ~]$ sudo systemctl start pcsd
[sudo] password di yanakozak:
[yanakozak@sanjunipero ~]$ sudo pcs cluster auth sanjunipero westworld
[sudo] password di yanakozak:
sanjunipero: Already authorized
westworld: Already authorized
```

4. Now we can actually create our cluster by adding the nodes - using the following line on the node we will use to manage the others:

```
sudo pcs cluster setup --name cluster_bello sanjunipero westworld
```

```
[root@sanjunipero corosync]# pcs cluster auth sanjunipero westworld
sanjunipero: Already authorized
westworld: Already authorized
[root@sanjunipero corosync]# pcs cluster setup --name cluster_bello sanjunipero westworld
Destroying cluster on nodes: sanjunipero, westworld...
sanjunipero: Stopping Cluster (pacemaker)...
westworld: Stopping Cluster (pacemaker)...
sanjunipero: Successfully destroyed cluster
westworld: Successfully destroyed cluster

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

Synchronizing pcsd certificates on nodes sanjunipero, westworld...
sanjunipero: Success
westworld: Success
Restarting pcsd on the nodes in order to reload the certificates...
westworld: Success
sanjunipero: Success
[root@sanjunipero corosync]# pcs cluster start --all
sanjunipero: Starting Cluster...
westworld: Starting Cluster...
```

5. At this point we are ready to start our cluster:

```
sudo pcs cluster start --all
```

You can check the status of your cluster with `pcs status`, which would output something similar to what is contained in Figure 10.

```
[root@sanjunipero corosync]# sudo pcs status corosync
```

Membership information

```
-----
Nodeid      Votes Name
    2         1 westworld
    1         1 sanjunipero (local)
```

Figure 9: Membership information.

```
[root@sanjunipero corosync]# pcs status
Cluster name: cluster_bello
WARNING: no stonith devices and stonith-enabled is not false
Stack: corosync
Current DC: sanjunipero (version 1.1.18-2.fc28.1-2b07d5c5a9) - partition with quorum
Last updated: Sun Oct 14 18:27:00 2018
Last change: Sun Oct 14 18:23:41 2018 by hacluster via crmd on sanjunipero

2 nodes configured
0 resources configured

Online: [ sanjunipero westworld ]

No resources

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

Figure 10: Checking status of the cluster.

```
[root@sanjunipero corosync]# sudo corosync-cmapctl | grep members
runtime.totem.pg.mrp.srp.members.1.config_version (u64) = 0
runtime.totem.pg.mrp.srp.members.1.ip (str) = r(0) ip(192.168.137.4)
runtime.totem.pg.mrp.srp.members.1.join_count (u32) = 1
runtime.totem.pg.mrp.srp.members.1.status (str) = joined
runtime.totem.pg.mrp.srp.members.2.config_version (u64) = 0
runtime.totem.pg.mrp.srp.members.2.ip (str) = r(0) ip(192.168.137.3)
runtime.totem.pg.mrp.srp.members.2.join_count (u32) = 1
runtime.totem.pg.mrp.srp.members.2.status (str) = joined
```

Figure 11: Info from Corosync object database.

## 6 Configuration of the cluster

Now we can check whether our configuration has any errors:

```
[root@sanjunipero corosync]# sudo crm_verify -L -V
error: unpack_resources: Resource start-up disabled since no STONITH resources have been defined
error: unpack_resources: Either configure some or disable STONITH with the stonith-enabled option
error: unpack_resources: NOTE: Clusters with shared data need STONITH to ensure data integrity
Errors found during check: config not valid
```

STONITH is a tool for isolation of the failed nodes that could disrupt the computer cluster. As our cluster is not too sophisticated we are better off using it in this experiment. So we will type:

```
sudo pcs property set stonith-enabled=false.
```

As by default the low quorum policy is enabled on HA clusters, which guarantees the availability of the service only when there is more than half of the nodes active simultaneously - in our case it would not let our cluster work, so we disable it too:

```
sudo pcs property set no-quorum-policy=ignore
```

### 6.1 Creating a virtual IP

Now we need an IP by which it will be possible to reach our service. We typed this:

```
sudo pcs resource create virtual_ip ocf:heartbeat:IPaddr2 ip=192.168.137.100
cidr_netmask=28 op monitor interval=30s
```

## 7 Configuring Apache webserver

1. As we have already installed `httpd`, now we change iptables to allow traffic through the port 80:

```
sudo iptables -I INPUT -p tcp -m state --state NEW -m tcp --dport 80 -j
ACCEPT
sudo service iptables save
```

2. Now we need to create a mechanism to test regularly whether our Apache service is responding on the active node: we will create a file `/etc/httpd/conf.d/serverstatus.conf` which will be queried constantly but won't be accessible from outside. The contents of the file are:

```
Listen 127.0.0.1:80
<Location /server-status>
SetHandler server-status
Order deny,allow
```

```

    Deny from all
    Allow from 127.0.0.1
</Location>

```

3. We use the next command, moreover, to prevent our nodes from listening all ports 80 - relative to each interface

```
sudo sed -i 's/Listen/#Listen/' /etc/httpd/conf/httpd.conf
```

4. A simple, static `/var/www/html/index.html` has been created on each node containing lines like these:

```

<html>
<h1>sanjunipero rules!<\h1>
<\html>

```

5. At this point we stop `httpd` service on both nodes:

```
sudo systemctl stop httpd
```

In fact our cluster will take care of managing this service on the cluster nodes

6. Indicate to the nodes to listen to the port 80 of the virtual IP:

```
echo "Listen 192.168.202.100:80"|sudo tee --append /etc/httpd/conf/httpd.conf
```

7. Finally we can create the very webservice resource from one of the nodes and from then on http will be managed by cluster, as all of the components that are needed to define it has been configured.:

```
sudo pcs resource create webserver ocf:heartbeat:apache configfile=/etc/httpd/
/conf/httpd.conf statusurl="http://localhost/server-status" op monitor inter-
val=1min
```

8. Add the following lines to be sure to find both resources (virtual IP and webserver) on the same node - as a cluster may apply load balancing by default and could have moved our services on two distinct machines

```
f sudo pcs constraint colocation add webserver virtual_ip INFINITY
```

9. Decide order of activation of our resources:

```
sudo pcs constraint order virtual_ip then webserver
```

The constraints that have been defined are illustrated in Figure 12.

```
[root@sanjunipero html]# sudo pcs constraint
Location Constraints:
Ordering Constraints:
    start virtual_ip then start webserver (kind:Mandatory)
Colocation Constraints:
    webserver with virtual_ip (score:INFINITY)
Ticket Constraints:
```

Figure 12: A list of the constraints relative to the cluster resources - that is virtual IP and webservice.

10. Now restart the cluster and check its status:

```
sudo pcs cluster stop --all && sudo pcs cluster start --all
sudo pcs status
```

```
[root@sanjunipero html]# sudo pcs cluster stop --all && sudo pcs cluster start --all
sanjunipero: Stopping Cluster (pacemaker)...
westworld: Stopping Cluster (pacemaker)...
sanjunipero: Stopping Cluster (corosync)...
westworld: Stopping Cluster (corosync)...
sanjunipero: Starting Cluster...
westworld: Starting Cluster...
[root@sanjunipero html]# sudo pcs status
Cluster name: cluster_bello
Stack: corosync
Current DC: sanjunipero (version 1.1.18-2.fc28.1-2b07d5c5a9) - partition with quorum
Last updated: Sun Oct 14 22:53:14 2018
Last change: Sun Oct 14 22:17:39 2018 by root via cibadmin on sanjunipero

2 nodes configured
2 resources configured

Online: [ sanjunipero westworld ]

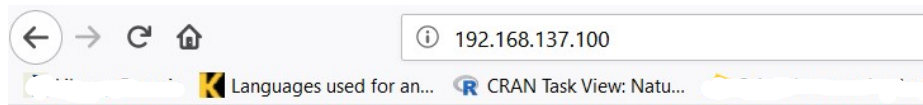
Full list of resources:

virtual_ip      (ocf::heartbeat:IPaddr2):      Started sanjunipero
webserver       (ocf::heartbeat:apache):      Started sanjunipero

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

Figure 13: Restarting cluster after having delegated to it Apache.

Figure 14 is instead what we should see if “navigate to” to **192.168.137.100** on our web browser:



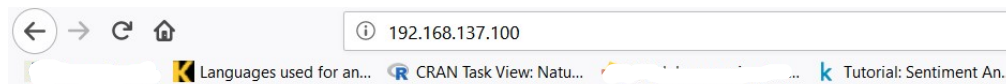
**sanjunipero rules!**

Figure 14: Connecting to our webservice from host machine.

Now we stop our **sanjunipero** node by writing:

```
sudo pcs cluster stop sanjunipero
```

and try to access another time our website. What we obtain is this:



**westworld is great but sanjunipero is much better!**

Figure 15: Accessing website after the active node has been stopped.

```

[root@westworld conf]# pcs status
Cluster name: cluster_bello
Stack: corosync
Current DC: westworld (version 1.1.18-2.fc28.1-2b07d5c5a9) - partition with quorum
Last updated: Sun Oct 14 23:24:15 2018
Last change: Sun Oct 14 22:17:39 2018 by root via cibadmin on sanjunipero

2 nodes configured
2 resources configured

Online: [ westworld ]
OFFLINE: [ sanjunipero ]

Full list of resources:

virtual_ip      (ocf::heartbeat:IPaddr2):      Started westworld
webserver       (ocf::heartbeat:apache):      Started westworld

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

```

Figure 16: Viewing cluster status from the newly activated, secondary node called *westworld*.

## 7.1 Configuring DRBD

DRBD can be seen as a network level RAID 1 (which implies mirroring - creation of exact copies - of our data on at least 2 distinct discs in order to assure reliability of the system). Steps to configure it are the following:

1. Create a new (virtual) disk of the same size on both of our virtual machines. We dedicated to it 512MB.

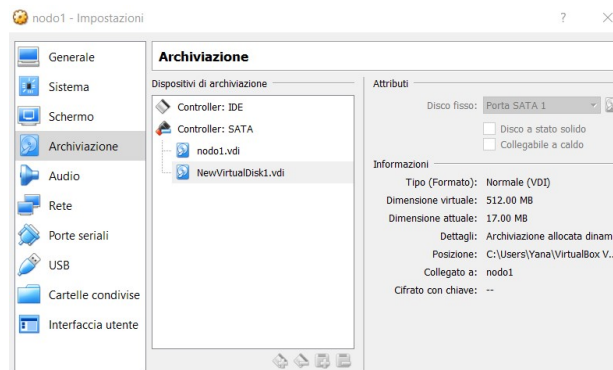


Figure 17: Creation of a separate disk , intended for DRBD.

2. Edit `/etc/drbd.conf` file on each node similarly to how it has been done in the following figure:



```

resource wwwdata {
    protocol C;
    meta-disk internal;
    device /dev/drbd1;
    syncer {
        verify-alg sha1;
    }
    net {
        allow-two-primaries;
    }
    on sanjunipero {
        disk /dev/sdb;
        address 192.168.137.4:7788;
    }
    on westworld {
        disk /dev/sdb;
        address 192.168.137.3:7788;
    }
}

```

Figure 18: Configuring DRBD via `/etc/drbd.conf` file.

3. begin
4. load the kernel module on each node typing:

```
sudo modprobe drbd
```

5. Create mirror device and bring it online:

```

sudo drbdadm create-md wwwdata
sudo drbdadm up wwwdata

```

6. Make one of the nodes primary:

```
sudo drbdadm -- --overwrite-data-of-peer primary wwwdata/0
```

```
[root@sanjunipero etc]# cat /proc/drbd
version: 8.4.10 (api:1/proto:86-101)
srcversion: A48A7BC6657B70D2F41F96E

1: cs:Connected ro:Primary/Secondary ds:UpToDate/UpToDate C r-----
   ns:524236 nr:0 dw:0 dr:524236 al:8 bm:0 lo:0 pe:0 ua:0 ap:0 ep:1 wo:f oos:0
```

Figure 19: The output of `cat /proc/drbd` after the initial set-up of our drbd resource.

7. Now we can format our disk on the primary node and mount it - we use the directory containing our static webpage so that both hosts will make available to the users the same "website":

```
sudo mkfs.ext4 /dev/drbd1
mount /dev/drbd1 /mnt sudo mkdir -p /var/www/html
echo "sanjunipero rules!" > /mnt/index.html umount /mnt
```

8. create a new cib file and type the following configurations - then apply the changes:

```
sudo pcs cluster cib add_drbd
sudo pcs -f add_drbd resource create WebData ocf:linbit:drbd drbd_resource=wwwdata
op monitor interval=60s
pcs -f add_drbd resource master WebDataClonec WebData master-max=1 master-node-max=1
clone-max=2 clone-node-max=1 notify=true
pcs cluster cib-push add_drbd
```

```
[root@sanjunipero /]# pcs cluster cib-push drbd_cfg
CIB updated
[root@sanjunipero /]# pcs status
Cluster name: cluster_bello
Stack: corosync
Current DC: westworld (version 1.1.18-2.fc28.1-2b07d5c5a9) - partition with
um
Last updated: Sun Oct 21 15:08:00 2018
Last change: Sun Oct 21 15:07:37 2018 by root via cibadmin on sanjunipero

2 nodes configured
4 resources configured

Online: [ sanjunipero westworld ]

Full list of resources:

virtual_ip      (ocf::heartbeat:IPaddr2):      Started sanjunipero
webserver       (ocf::heartbeat:apache):       Started sanjunipero
Master/Slave Set: WebDataClone [WebData]
Masters: [ sanjunipero ]
Slaves: [ westworld ]
```

Figure 20: Output of `pcs status` after integrating drbd resource inside cluster.

9. Now we create a filesystem resource on our cluster:

```
pcs cluster cib add_fs
pcs -f add_fs resource create WebFS Filesystem device="/dev/drbd1" directory
="/var/www/html" fstype="ext4"
pcs -f add_fs constraint colocation add WebFS WebDataClone INFINITY with-
rsc-role=Master
pcs -f add_fs constraint order promote WebDataClone then start WebFS
pcs -f add_fs constraint colocation add webserver WebFS INFINITY
pcs -f add_fs constraint order WebFS then webserver pcs cluster cib-push
add_fs
```

```
[root@sanjunipero by-res]# pcs status
Cluster name: cluster_bello
Stack: corosync
Current DC: sanjunipero (version 1.1.18-2.fc28.1-2b07d5c5a9) - partition with quorum
Last updated: Sun Oct 21 18:28:32 2018
Last change: Sun Oct 21 18:28:18 2018 by root via cibadmin on sanjunipero

2 nodes configured
5 resources configured

Online: [ sanjunipero westworld ]

Full list of resources:

virtual_ip      (ocf::heartbeat:IPaddr2):      Started sanjunipero
webserver       (ocf::heartbeat:apache):         Started sanjunipero
Master/Slave Set: WebDataClone [WebData]
    Masters: [ sanjunipero ]
    Slaves: [ westworld ]
WebFS   (ocf::heartbeat:Filesystem):  Started sanjunipero

Daemon Status:
corosync: active/enabled
pacemaker: active/enabled
pcsdd: active/enabled
```

Figure 21: Running `pcs status` after creation of filesystem resource.

## 8 Conclusion

A very simple, two-node HA-cluster has been realized thanks to open source software and tools such as Pacemaker, Corosync and DRBD, apart from Apache and VirtualBox and Fedora. It was interesting to create it on the virtualized machines, which would host a website (only one machine at a time would do it and the other would substitute it in case of failure). We have seen also that all of these tools are available for any Linux distribution, so there is a wide choice of operating systems that can benefit from HA mechanisms.