===== Basics =====

I’m currently fiddling around with high availability on Linux. I haven’t done this before, but a colleague of mine pointed me towards [[http://clusterlabs.org/|Pacemaker]] together with [[http://corosync.github.io/corosync/|Corosync]] (to keep things short: corosync keeps the cluster nodes in sync and pacemaker stops/starts/monitors services on the nodes). I’d like to walk you through a fairly complex example of configuring a clustered DHCP-Server. This setup is done in Ubuntu. Other distributions vary in package management and paths.

I’m assuming the following setup:

* two nodes (physical or virtual machines) called „nodeone“ and „nodetwo“
* eth0 – management interface for you to talk to them via ssh
* eth1 – HA interface, that is used for HA communication. I’m assuming the IPs „10.10.100.1/24“ and „10.10.100.2/24“ respectively
* eth2 – An interface that points towards the DHCP clients.
* They have a static network configuration. I’m assuming „127.0.1.10/32“ on both nodes.
* They have a DHCP network. I’m assuming „10.10.101.0/24“ with the virtual ip address „10.10.101.1“, that is shared between the two nodes.

The configuration is actually quite simple. You first install the needed packages using your package manager. For Ubuntu that would be

apt-get install pacemaker corosync

After that you’ll have to alter the settings in /etc/corosync/corosync.conf. In my case, this was just correcting the „bindnetaddr“ of the totem/interface-configuration item to the HA-network interface mentioned before.

Do the same on the second node. After that start corosync using

/etc/init.d/corosync start

Pacemaker should be started automatically. Now your basic cluster already works (you can use „crm_mon“ to look at your cluster).

But of course, your cluster doesn’t do anything, because he doesn’t know which things are important to have on the other side once one node drops off. This thing is called „resources“ and has to be configured into the pacemaker instance. There is a nice command line tool called „crm“ for this.

The first thing you have to do is to disable two actually very useful features of corosync/pacemaker:

* Disable STONITH. STONITH (meaning „Shoot The Other Node In The Head“) is a method of actively destroying the other node when taking over its services. This is accomplished by methods like IPMI, which can actually shut down the physical (or virtual) machine. This would be outside the scope of this post.
* Set the quorum policy to „ignore“. Normally a cluster only containing one active node isn’t called a cluster. There always have to be a minimum of two nodes or a „quorum“ of nodes that hold the cluster together. We’re planning on creating just an active/passive cluster with two nodes. When one node goes down, the other node should take over and be active. So we have to deconfigure corosync’s quorum-methods.

Do this by issuing

crm configure property stonith-enabled=false
crm configure property no-quorum-policy=ignore

===== Resources =====

Now, let’s get to the interesting part: The resources. Pacemaker calls services, mount points, network interfaces that the two nodes exchange „resources“. So we have to define our needed resources.

Let’s step back a bit. We want a DHCP-cluster. What does that imply? A DHCP-server basically consists of:

* an IP-Address (obviously)
* a configuration file
* a leases-file

Now, the dhcp-configuration file is rather static. We’ll just configure it on one node and copy it over to the other node. The leases-file is very dynamic as the DHCP-server changes it roughly every time a client speaks to it. So we need to share this file between the node. This calls for a cluster filesystem! I’d suggest „drbd“ here, because it’s quite fast and easy to setup.

==== DRBD ====

[[http://www.drbd.org|DRBD]] (short for „Distributed Replicated Block Device“) is – how the

vendor calls it himself – „Raid-1“ over the network. It keeps two block devices of two different network nodes in sync and offers a block device itself on the primary node. To setup, first install the package:

apt-get install drbd8-utils

Then start drbd:

/etc/init.d/drbd start

I assume, that you have an identical sized (and best: named) block device on both systems, let’s call it /dev/sdb1. To use drbd you have to setup a so-called „resource“ with these block devices. Let’s call that resource „r0“.

To configure it, edit a file named „r0.res“ inside /etc/drbd.d and fill it with this configuration:

```
resource r0 {
```

disk {

fencing resource-only;

}

net {

allow-two-primaries;

}

handlers {
fence-peer "/usr/lib/drbd/crm-fence-peer.sh";
after-resync-target "/usr/lib/drbd/crm-unfence-peer.sh";
}

on nodeone {

device /dev/drbd1;
disk /dev/sdb1;
address 10.10.100.1:7789;
meta-disk internal;

}

on nodetwo {

device /dev/drbd1;
disk /dev/sdb1;
address 10.10.100.2:7789;
meta-disk internal;

}

}

Okay, that's a mouthful. For now, just concentrate on the two „on nodeone" and „on nodetwo" settings. They just tell drbd the following:

* **device**: The device node, drbd should create
* **disk**: The block device, that is synced between the two nodes
* **address**: The IP-Address of the node
* **meta-disk internal**: Don't use any fancy methods for storing meta data, just use internal files on both nodes

Now, **on every node** do the following:

drbdadm create-md r0
drbdadm up r0

That will create the metadata, prepare /dev/sdb1 for the sync process and finally do the initial sync. After that monitor /proc/drbd and wait for the initial sync to finish. (That can take quite a long time depending on how big your device is).

Afterwards, make one node the „primary" node. That is the node, that can actually write on the device:

drbdadm primary r0

On this node you now can create the filesystem on the synced block device:

mkfs.ext3 /dev/drbd1

After that, again, watch /proc/drbd and wait for any synchronization to complete.

You’re done. You have created a synced block device on the two nodes. You can mount /dev/drbd1 – or the memorable /dev/drbd/by-res/r0 – on any mountpoint and write data to it. If you want to test the setup, make the currently primary node secondary by doing a

drbdadm secondary r0

and issuing

drbdadm primary r0

on the former secondary node. Now, mount the drbd-device. The contents should be the same on both nodes. Go ahead, create files, edit them and see, wether this thing works.

==== Placing the dhcp leases file ====

After we have a working drbd configuration, we can use the cluster device to save our leases to. After you have installed the DHCP-server by doing

apt-get install isc-dhcp-server

stop it using

service isc-dhcp-server stop

and copy the directory /var/lib/dhcp to the cluster device. Please check, that the permissions and the ownership are copied, too.

Also check, that the „dhcpd“ user and group on both nodes have the same userid and groupid respectively!

To point the DHCP-server to the new location, edit the dhcpd.conf in /etc/dhcp and add the parameter

lease-file-name „„;

to it.

I guess you saw, that you need to have a fixed address on eth2 and that the actual network (10.10.101.0/24) is shared between the nodes. That is so, because we want pacemaker to just switch over an IP address, not disable/enable network devices. This, however, requires a „shared-network“ configuration inside the DHCP-server.

So before you start actually configuring the DHCP-server add the following snippet to your DHCP-configuration file:

```
shared-network eth2 {
```

subnet 127.0.1.10 netmask 255.255.255.255 {
}
subnet 10.10.101.0 netmask 255.255.255.0 {
}

}

(Please note, that „eth2″ is merely a name here)

After that, go ahead and configure the DHCP-server according to your needs. When you're finished install the DHCP-server on the other node and copy over the configuration file. **Don't** start the server on any node!

==== Making Pacemaker switch IP addresses ====

Now, we actually configure a resource! Do the following:

crm configure primitive testnet ocf:heartbeat:IPaddr2 params ip="10.10.101.1″ cidr_netmask="24″ nic="eth2"

Let's look at the line:

* **crm configure** – enter the configuration mode of the crm program
* **primitive** – Add a „primitive", a very basic resource
* **testnet** – The name of our resource
* **ocf:heartbeat:IPaddr2** – The „resource agent" of our resource. That is actually a shell script handling actions for pacemaker. The scripts are organized using vendor („ocf") and agent-groups („heartbeat") and the actual name of the script („IPaddr2")
* **params** – Now come the params!
* **ip** – The IP to share among the cluster nodes
* **cidr_netmask** – The netmask of the IP in CIDR notation
* **nic** – The network interface

That's it. Now one of your nodes holds the ip address 10.10.101.1. (use „ip addr list" on both nodes to see which). If you stop corosync one that node, the other node should automatically

take the IP.

==== Pacemaker and drbd ====

Now comes the complicated part. DRBD works quite different than pacemaker. It holds a constant stream of synchronization between the two nodes and only lets primary nodes access the block devices. So it cannot just be stopped on the node going down and started on the other node. For this, two configurations are needed inside DRBD:

* DRBD must allow two primary nodes. („allow-two-primaries“)
* DRBD must use fencing

The first one is quite simple to understand as it allows two primary nodes while corosync moves the resources from one node to another. The second one is needed, when two nodes are actually working. In this scenario, the slave node has to be „fenced off“ drbd, so that commands coming from that node are simply ignored. That is configured using the „fencing resource-only“ statement and the handlers-group in the resource configuration we already made.

To include drbd-switching into pacemaker we have to not only create a primitive resource, but also a master/slave-resource out of the primitive resource:

crm configure primitive drbd-r0 ocf:linbit:drbd params drbd_resource=“r0″ op monitor interval=“29s“ role=“Master“ op monitor interval=“31s“ role=“Slave“
crm configure ms drbd-r0-master drbd-r0 meta master-max=“1″ master-node-max=“1″ clone-max=“2″ clone-node-max=“1″

Let’s look at that:

* **ocf:linbit:drbd** – That’s the resource agent for drbd. There is also another one, that is deprecated
* **drbd_resource** – The name of our drbd resource
* **monitor interval(…)** – These two things control the check interval when the node is Master or Slave. These generally just doesn’t have to be the same values
* **ms** – We’re adding a „Master/Slave“ resource (instead of a „primitive“)
* **drbd-r0-master** – The name of our ms-resource
* **drbd-r0** – The name of our primitive, that is either master or slave
* **meta (…)** – Additional configuration for master-slave, telling them, that there has to be exactly one master.

Let’s add another line:

crm configure colocation drbd_on_net inf: testnet drbd

That line tells pacemaker, that drbd and testnet should always run **both** on the currently active node (specified by „inf:“).

That should do it. If you look at /proc/drbd the currently active node should be „primary“, while the other one should be „secondary“. If you switch nodes and look at /proc/drbd on the now active node, the output should be the same (the first entry always refers to the machine you’re currently on).

==== Mounting ====

But what good would drbd do without a mounted device? So let’s also add a mount-resource to pacemaker:

crm configure primitive fs_data ocf:heartbeat:Filesystem params device=“/dev/drbd/by-res/r0″ directory=“/data“ fstype=“ext3″

Here we let ocf:hearbeat:Filesystem mount the device /dev/drbd/by-res/r0 to /data using the ext3 filesystem on the active node (change the values to match your setup). But not let’s forget this one here:

crm configure colocation fs_on_drbd inf: drbd-r0-master:Master fs_data

The mount-resource obviously should only mount the device, when the node is drbd master! Also, add this one:

crm configure order fs_after_drbd inf: drbd-r0-master:promote fs_data:start

This tells pacemaker to first „promote“ the drbd resource (make the active node primary in terms of drbd) and after that „start“ the mount-resource (mount it).

These two things are crucial or the cluster would go totaly nuts.

==== A clustered DHCP-Server ====

Okay, we have the IP-address, the leases-file using a mounted drbd-resource. Now the DHCP-Server.

„That’s easy“ I hear you say. „Just stop the dhcp-server on the broken node and start it on

the active node!“

Yyyes. That totally makes sense. The only thing is, that there’s no such resource agent for dhcp. There actually is a „isc-dhcp-server“ agent which you can find in the „lsb“-group, but that doesn’t work, because the lsb-group just mimics the scripts in /etc/init.d as resource agents and that (at least on Ubuntu) doesn’t work.

But don’t worry, I got you covered. It’s actually quite ease to write your own resource agents. They’re basically shell scripts, that have to implement certain calls. I won’t cover it here, but you can read about it [[http://www.linux-ha.org/wiki/OCF_Resource_Agents|here]].

I’ll just post my resource agent:

```
#!/bin/bash
#
#
# OCF Resource Agent compliant dhcp resource script.
#

#######################################################################
# Initialization:

: ${OCF_FUNCTIONS_DIR=${OCF_ROOT}/resource.d/heartbeat}
. ${OCF_FUNCTIONS_DIR}/.ocf-shellfuncs

# The passed in OCF_CRM_meta_notify_* environment
# is not reliably with pacemaker up to at least
# 1.0.10 and 1.1.4. It should be fixed later.
# Until that is fixed, the "self-outdating feature" would base its actions on
# wrong information, and possibly not outdate when it should, or, even worse,
# outdate the last remaining valid copy.
# Disable.
OCF_RESKEY_stop_outdates_secondary_default="false"

: ${OCF_RESKEY_drbdconf:=${OCF_RESKEY_drbdconf_default}}
:
${OCF_RESKEY_stop_outdates_secondary:=${OCF_RESKEY_stop_outdates_secondary_default
}}
```

meta_data() {
cat <

1.3

This resource agent manages an ISC dhcp server by stopping and starting it via its init script.

The DHCP server should position its leases-file on a cluster shared filesystem (e.g. drbd) to work properly in a cluster.

Manages a ISC dhcp server

The service name of the DHCP-server.
(Defaults to isc-dhcp-server)

DHCP-service name

END
}

dhcp_usage() {
cat <&1 >/dev/null

sleep 4

dhcp_status=`dhcp_served`

if ["$dhcp_status" = "ok"]; then

```
exit $OCF_SUCCESS
fi

exit $OCF_NOT_RUNNING

}

dhcp_stop() {

local dhcp_status=`dhcp_served`

if [ "$dhcp_status" = "no" ]; then
exit $OCF_SUCCESS
fi

service $OCF_RESKEY_service stop 2>&1 >/dev/null

dhcp_status=`dhcp_served`

if [ "$dhcp_status" = "no" ]; then
exit $OCF_SUCCESS
fi

exit $OCF_ERR_GENERIC

}

dhcp_served() {

export LANG=C

local dhcp_status=`service ${OCF_RESKEY_service} status 2>&1`

if [ $? -ne 0 ]; then
ocf_log err "Service $OCF_RESKEY_service not registered."
exit $OCF_ERR_CONFIGURED
fi

echo $dhcp_status | grep "start/running" 2>&1 >/dev/null

if [ $? -eq 0 ]; then
echo "ok"
```

```
else
echo "no"
fi

return 0

}

dhcp_monitor() {

local dhcp_status=`dhcp_served`

case $dhcp_status in
ok)
return $OCF_SUCCESS
;;
no)
return $OCF_NOT_RUNNING
;;
*)
return $OCF_ERR_GENERIC
;;
esac

}

if [ X"$OCF_RESKEY_service" = "X" ]; then
OCF_RESKEY_service="isc-dhcp-server"
fi

case $__OCF_ACTION in
meta-data) meta_data
exit $OCF_SUCCESS
;;
usage|help) dhcp_usage
exit $OCF_SUCCESS
;;
esac
```

```
case $__OCF_ACTION in
start) dhcp_start
;;
stop) dhcp_stop
;;
status) dhcp_status=`dhcp_served`
if [ $dhcp_status = "ok" ]; then
echo "running"
exit $OCF_SUCCESS
else
echo "stopped"
exit $OCF_NOT_RUNNING
fi
;;
monitor) dhcp_monitor
;;
validate-all) ;;
*) dhcp_usage
exit $OCF_ERR_UNIMPLEMENTED
;;
esac
```

Create a directory under /usr/lib/ocf/resource.d (let's say „dieploegers"), put the code there in a script called „dhcp" and make the script executable.

Now you have a new resource agent at your hands. Do the following:

```
crm configure primitive dhcp ocf:dieploegers:dhcp
crm configure colocation dhcp_on_net inf: testnet dhcp
crm configure colocation dhcp_on_fs inf: fs_data dhcp
crm configure order dhcp_after_fs inf: fs_data:start dhcp:start
```

This obviously tells pacemaker to add a resource based on our new resource agent and let it reside on the active node only when the testnet and fs_data resources are available. And before starting it, mount the data mountpoint, that holds the dhcp leases.

That was the final step. You cluster should now move all of your resources from one node to the other and respect the order the resources should start in.