Table of Contents

Cluster Creation for RHEL/CentOS 7

As mentioned before, *RedHat* has deprecated *CMAN/Ricci* on the latest version of the *Enterprise Linux*, and now the default cluster solution is based on *Pacemaker*.

To manage the cluster nodes, we will use *PCS*. This allows us to have a single interface to manage all cluster nodes. By installing the necessary packages, *Yum* also created a user, *hacluster*, which can be used together with *PCS* to do the configuration of the cluster nodes. Before we can use *PCS*, we need to configure public key authentication or give the user a password on both nodes:

```
[root@onmssrv01 ~]# passwd hacluster
Changing password for user hacluster.
New password:
BAD PASSWORD: The password is shorter than 8 characters
Retype new password:
passwd: all authentication tokens updated successfully.

[root@onmssrv02 ~]# passwd hacluster
Changing password for user hacluster.
New password:
BAD PASSWORD: The password is shorter than 8 characters
Retype new password:
passwd: all authentication tokens updated successfully.
```

Next, start the pcsd service on both nodes:

```
[root@onmssrv01 ~]# systemctl start pcsd
[root@onmssrv02 ~]# systemctl start pcsd
```

Since we will configure all nodes from one point, we need to authenticate on all nodes before we are allowed to change the configuration. Use the previously configured *hacluster* user and password to do this.

```
[root@onmssrv01 ~]# pcs cluster auth onmssrv01 onmssrv02
 Username: hacluster
 Password:
 onmssrv01: Authorized
 onmssrv02: Authorized
```

From here, we can control the cluster by using *PCS* from *onmssrv01*. It's no longer required to repeat all commands on both nodes (imagine you need to configure a 100-node cluster without automation).

Cluster Creation

We'll start by adding both nodes to a cluster named cluster_onms:

```
[root@onmssrv01 ~]# pcs cluster setup --name cluster_onms onmssrv01 onmssrv02
Shutting down pacemaker/corosync services...
Redirecting to /bin/systemctl stop  pacemaker.service
Redirecting to /bin/systemctl stop  corosync.service
Killing any remaining services...
Removing all cluster configuration files...
onmssrv01: Succeeded
onmssrv02: Succeeded
```

The above command creates the cluster node configuration in `/etc/corosync/corosync.conf`. The syntax in that file is quite readable in case you would like to automate/script this.

WARNING	I've used the short names for the nodes, if you want to use the *FQDN* keep in mind that all the commands shown bellow should use same method when referencing cluster nodes.

After creating the cluster and adding nodes to it, we can start it. The cluster won't do a lot yet since we didn't configure any resources.

```
[root@onmssrv01 ~]# pcs cluster start --all
onmssrv02: Starting Cluster...
onmssrv01: Starting Cluster...
```

You could also start the *pacemaker* and *corosync* services on both nodes (as will happen at boot time) to accomplish this.

To check the status of the cluster after starting it:

```
[root@onmssrv01 ~]# pcs cluster status
Cluster Status:
 Last updated: Wed Jul 29 19:40:41 2015
 Last change: Wed Jul 29 19:40:34 2015
 Current DC: NONE
 2 Nodes configured
 0 Resources configured

PCSD Status:
  onmssrv01: Online
  onmssrv02: Online
```

To check the status of the nodes in the cluster:

```
[root@onmssrv01 ~]# pcs status nodes
Pacemaker Nodes:
 Online: onmssrv01 onmssrv02
 Standby:
 Offline:
```

IMPORTANT	in case they appear offline is because *stonish* is enabled (more on this below).

```
[root@onmssrv01 ~]# 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(192.168.205.151)
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(192.168.205.152)
runtime.totem.pg.mrp.srp.members.2.join_count (u32) = 2
runtime.totem.pg.mrp.srp.members.2.status (str) = joined

[root@onmssrv01 ~]# pcs status corosync

Membership information
----------------------
    Nodeid      Votes Name
         1          1 onmssrv01 (local)
         2          1 onmssrv02
```

To check the configuration for errors, and there still are some:

```
[root@onmssrv01 ~]# 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
```

The above message tells us that there still is an error regarding *STONITH*, which is a mechanism to ensure that you don't end up with two nodes that both think they are active and claim to be the service and virtual IP owner, also called a split brain situation.

For now, we'll just disable the *STONITH* option, but will cover it later.

```
[root@onmssrv01 ~]# pcs property set stonith-enabled=false
```

While configuring the behavior of the cluster, we can also configure the quorum settings. The quorum describes the minimum number of nodes in the cluster that need to be active in order for the cluster to be available. This can be handy in a situation where a lot of nodes provide simultaneous computing power. When the number of available nodes is too low, it's better to stop the cluster rather than deliver a non-working service. By default, the quorum is considered too low if the total number of nodes is smaller than twice the number of active nodes. For a 2 node cluster that means that both nodes need to be available in order for the cluster to be available. In our case this would completely destroy the purpose of the cluster.

At this point the nodes should appear online:

```
[root@onmssrv01 ~]# pcs status nodes
Pacemaker Nodes:
 Online: onmssrv01 onmssrv02
 Standby:
 Offline:
```

To ignore a low quorum:

```
[root@onmssrv01 ~]# pcs property set no-quorum-policy=ignore

[root@onmssrv01 ~]# pcs property
Cluster Properties:
 cluster-infrastructure: corosync
 cluster-name: cluster_onms
 dc-version: 1.1.12-a14efad
 have-watchdog: false
 no-quorum-policy: ignore
 stonith-enabled: false
```

The cluster resources we're going to add are the following:

The floating IP Address * A shared filesystem for `/opt/opennms/etc` * A shared filesystem for `/var/opennms` * A shared filesystem for `/etc/pgpool-II` * The init script for *pgpool-II* * The init script for *OpenNMS*

One of the most common elements of a cluster is a set of resources that need to be located together, start sequentially, and stop in the reverse order. To simplify this configuration, Pacemaker supports the concept of groups (this is similar to the cluster service in *CMAN*).

The fundamental properties of a group are as follows:

There is no limit to the number of resources a group can contain. Resources are started in the order in which you specify them. Resources are stopped in the reverse order in which you specify them. If a resource in the group cannot run anywhere, then no resource specified after that resource is allowed to run.

To simplify the configuration each resource creation instruction contains the group on which the resource should be added (in this case, onms_app). If the group doesn't exist, it will be created automatically.

Create the virtual IP is the IP address that which will be contacted to reach the services (the OpenNMS application in our case):

```
[root@onmssrv01 ~]# pcs resource create virtual_ip ocf:heartbeat:IPaddr2 \
ip=192.168.205.150 cidr_netmask=32 \
op monitor interval=30s on-fail=standby \
--group onms_app meta target-role="Started" migration-threshold="1"
```

Create the cluster resources for the shared file systems:

```
[root@onmssrv01 ~]# pcs resource create onms_etc ocf:heartbeat:Filesystem \
device="nfssrv01:/opt/opennms/etc" directory="/opt/opennms/etc" fstype="nfs" \
op monitor interval=30s on-fail=standby \
--group onms_app meta target-role="Started" migration-threshold="1"

[root@onmssrv01 ~]# pcs resource create onms_var ocf:heartbeat:Filesystem \
device="nfssrv01:/opt/opennms/share" directory="/var/opennms" fstype="nfs" \
op monitor interval=30s on-fail=standby \
--group onms_app meta target-role="Started" migration-threshold="1"

[root@onmssrv01 ~]# pcs resource create pgpool_etc ocf:heartbeat:Filesystem \
device="nfssrv01:/opt/opennms/pgpool" directory="/etc/pgpool-II" fstype="nfs" \
op monitor interval=30s on-fail=standby \
--group onms_app meta target-role="Started" migration-threshold="1"
```

IMPORTANT	if you have issues with the *NFS* permissions for the *pgpool-II* configuration directory, do not add a resource for it.

Create the cluster resources for the application using *systemd*:

```
[root@onmssrv01 ~]# pcs resource create pgpool_bin systemd:pgpool \
op monitor interval=30s on-fail=standby \
--group onms_app meta target-role="Started" migration-threshold="1"

[root@onmssrv01 ~]# pcs resource create onms_bin systemd:opennms \
op start timeout=180s \
op stop timeout=180s \
op monitor interval=60s on-fail=standby \
--group onms_app meta target-role="Started" migration-threshold="1"
```

The value for the start timeout for the *opennms* resource, must be consistent with the value configured on `/opt/opennms/etc/opennms.conf` for `START_TIMEOUT`, and also with the value configured on `/lib/systemd/system/opennms.service` for `TimeoutStartSec`.

Because all the cluster resources belong to the same group, all the resources will always run on the same machine. If something wrong happens with one of them, all the resources will be moved to another cluster node automatically.

All the resources has the following two meta options: `target-role` which is configured to be `Started`, and `migration-threshold`, which is configured to be `1`. That means, all the resources must be running at the same time on the same node, and if one resource fails once (that's what `1` means for `migration-threshold`), all of them will be migrated to another cluster node. This is basically the same behavior you see on *CMAN* for *RHEL/CentOS 6*.

You can tune the migration threshold to be more than `1`, and *Pacemaker* will try to restart the service by that amount of times before migrate them to another node.

Now, it is important to understand that with the creation of the resource group, a location constraint will be added automatically:

```
[root@onmssrv01 ~]# pcs constraint show --full
Location Constraints:
  Resource: onms_app
    Enabled on: onmssrv01 (score:INFINITY) (role: Started) (id:cli-prefer-onms_app)
Ordering Constraints:
Colocation Constraints:
```

That means, if the resource are migrated to *onmssrv02* because *onmssrv01* is dead, when *onmssrv01* is back, the resources will be moved back to *onmssrv01*, because this node is *preferred*.

You can check the status of the cluster with the `pcs status` command:

```
[root@onmssrv01 ~]# pcs status
Cluster name: cluster_onms
Last updated: Mon Jul 20 23:04:15 2015
Last change: Mon Jul 20 23:01:24 2015
Stack: corosync
Current DC: onmssrv11 (1) - partition with quorum
Version: 1.1.12-a14efad
2 Nodes configured
6 Resources configured

Online: [ onmssrv01 onmssrv02 ]

Full list of resources:

 Resource Group: onms_app
     virtual_ip(ocf::heartbeat:IPaddr2):Started onmssrv01
     onms_etc(ocf::heartbeat:Filesystem):Started onmssrv01
     onms_var(ocf::heartbeat:Filesystem):Started onmssrv01
     pgpool_etc(ocf::heartbeat:Filesystem):Started onmssrv01
     pgpool_bin(systemd:pgpool):Started onmssrv01
     onms_bin(systemd:opennms):Started onmssrv01

PCSD Status:
  onmssrv11: Online
  onmssrv12: Online

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

IMPORTANT *Pacemaker* doesn't show temporary states on the resources. It only shows *Started* or *Stopped*. If you see the *OpenNMS* resource *Stopped*, check with the `ps` command to see if it is running, as probably it is still starting.

As you can see, all the resources are running on the same node. At this time, the *OpenNMS* application must be reachable through the virtual IP address.

In order to see the configuration of the cluster, you can use the following command:

```
[root@onmssrv01 ~]# pcs config show
Cluster Name: cluster_onms
Corosync Nodes:
 onmssrv01 onmssrv02
Pacemaker Nodes:
 onmssrv01 onmssrv02

Resources:
 Group: onms_app
  Resource: virtual_ip (class=ocf provider=heartbeat type=IPaddr2)
   Attributes: ip=192.168.205.150 cidr_netmask=32
   Meta Attrs: target-role=Started migration-threshold=1
   Operations: start interval=0s timeout=20s (virtual_ip-start-timeout-20s)
               stop interval=0s timeout=20s (virtual_ip-stop-timeout-20s)
               monitor interval=30s on-fail=standby (virtual_ip-monitor-interval-30s)
  Resource: onms_etc (class=ocf provider=heartbeat type=Filesystem)
   Attributes: device=nfssrv01:/opt/opennms/etc directory=/opt/opennms/etc fstype=nfs
   Meta Attrs: target-role=Started migration-threshold=1
   Operations: start interval=0s timeout=60 (onms_etc-start-timeout-60)
               stop interval=0s timeout=60 (onms_etc-stop-timeout-60)
               monitor interval=30s on-fail=standby (onms_etc-monitor-interval-30s)
  Resource: onms_var (class=ocf provider=heartbeat type=Filesystem)
   Attributes: device=nfssrv01:/opt/opennms/share directory=/var/opennms fstype=nfs
   Meta Attrs: target-role=Started migration-threshold=1
   Operations: start interval=0s timeout=60 (onms_var-start-timeout-60)
               stop interval=0s timeout=60 (onms_var-stop-timeout-60)
               monitor interval=30s on-fail=standby (onms_var-monitor-interval-30s)
  Resource: pgpool_etc (class=ocf provider=heartbeat type=Filesystem)
   Attributes: device=nfssrv01:/opt/opennms/pgpool directory=/etc/pgpool-II fstype=nfs
   Meta Attrs: target-role=Started migration-threshold=1
   Operations: start interval=0s timeout=60 (pgpool_etc-start-timeout-60)
               stop interval=0s timeout=60 (pgpool_etc-stop-timeout-60)
               monitor interval=30s on-fail=standby (pgpool_etc-monitor-interval-30s)
  Resource: pgpool_bin (class=systemd type=pgpool)
   Meta Attrs: target-role=Started migration-threshold=1
   Operations: monitor interval=30s on-fail=standby (pgpool_bin-monitor-interval-30s)
  Resource: onms_bin (class=systemd type=opennms)
   Meta Attrs: target-role=Started migration-threshold=1
   Operations: start interval=0s timeout=180s (onms_bin-start-timeout-180s)
               stop interval=0s timeout=180s (onms_bin-stop-timeout-180s)
               monitor interval=60s on-fail=standby (onms_bin-monitor-interval-60s)

Stonith Devices:
Fencing Levels:

Location Constraints:
Ordering Constraints:
Colocation Constraints:

Cluster Properties:
 cluster-infrastructure: corosync
 cluster-name: cluster_onms
 dc-version: 1.1.12-a14efad
 have-watchdog: false
 no-quorum-policy: ignore
 stonith-enabled: false
```

Finally, enable the cluster services on both *OpenNMS* servers:

```
[root@onmssrv01 ~]# systemctl enable pcsd
[root@onmssrv01 ~]# systemctl enable corosync
[root@onmssrv01 ~]# systemctl enable pacemaker

[root@onmssrv02 ~]# systemctl enable pcsd
[root@onmssrv02 ~]# systemctl enable corosync
[root@onmssrv02 ~]# systemctl enable pacemaker
```

After enabling the services, you should see that the status of the daemons is updated when running `pcs status`:

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

In case it is not obvious, the `pcs status` can be executed from any cluster member.

Test Failover

In order to understand how to perform a failover test, it is recommended to read the information posted on the following link:

https://www.hastexo.com/blogs/martin/2012/07/11/failover-testing-some-technical-background

One way to test failover is by manually stopping one the critical resources on the active node, the obvious one is the *OpenNMS* application:

On *RHEL/CentOS 6*:

```
[root@onmssrv01 ~]# service opennms stop
```

On *RHEL/CentOS 7*:

```
[root@onmssrv01 ~]# systemctl stop opennms
```

That will trigger the cluster failover operation which is move the resources defined on the `onms_app` group to another cluster node.

```
[root@onmssrv01 ~]# pcs status
Cluster name: cluster_onms
...
 Resource Group: onms_app
     virtual_ip(ocf::heartbeat:IPaddr2):Started onmssrv02
     onms_etc(ocf::heartbeat:Filesystem):Started onmssrv02
     onms_var(ocf::heartbeat:Filesystem):Started onmssrv02
     pgpool_etc(ocf::heartbeat:Filesystem):Started onmssrv02
     pgpool_bin(systemd:pgpool):Started onmssrv02
     onms_bin(systemd:opennms):Started onmssrv02
```

WARNING	There's always going to be a small gap when doing a failover on the *DB* cluster or the application cluster, so during that time, the application could miss some *DB* transactions or external events (like *SNMP Traps* or *Syslog* messages).

As you can see, it is not a good idea to manually stop OpenNMS on a cluster. Later, I'll mention how to properly restart OpenNMS within a cluster.

Another way to test the failover is to stop the cluster services on the node on which the resource group is running:

```
[root@onmssrv01 ~]# pcs cluster stop onmssrv02
```

Keep in mind that the `onms_app` group will continue running on *onmssrv02*, until it fails or an administrator manually move them to another node.

To move the resource group to a different node:

```
[root@onmssrv01 ~]# pcs resource move onms_app onmssrv02
```

If you see failed actions when running `pcs status`, you can use the `pcs resource cleanup` to try to auto-fix the problem (in case there are any).

WebUI for Managing Cluster (pcsd)

Pacemaker provides a *WebUI* for the *PCS* command to configure and manage the cluster.

Keep in mind that this is totally optional, and not necessary. In order to use *pcsd WebUI*, make sure the `pcs cluster auth` command has been executed on all the cluster members, using the *hacluster* user created.

```
[root@onmssrv01 ~]# pcs cluster auth onmssrv01 onmssrv02
[root@onmssrv02 ~]# pcs cluster auth onmssrv01 onmssrv02
```

Then, use your browser and point them to any of following *URLs*, using the *hacluster* user:

```
https://onmssrv01:2224/
https://onmssrv01:2224/
```

The first time you open the *WebUI*, you should click on *Add Existing* and use the name of one of the cluster nodes.

For more information, follow this link:

https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/7/html/High_Availability_Add-On_Reference/ch-pcsd-HAAR.html

Restart OpenNMS

Because *OpenNMS* is part of a cluster service, the standard way to start, stop and restart *OpenNMS* cannot be used. Otherwise, the cluster will be confused.

The following command must be used from the active node to restart *OpenNMS* on the same cluster node:

```
[root@onmssrv01 ~]# pcs resource restart onms_app
```

This operation could take a few minutes, as it will trigger the stop/start process on each resource on the appropriate order. In other words, the resources of a group will be started on the same order they have been added to the group, and will be stopped on the reverse order.

The same command, with different parameters, can be used to temporarily disable the service or force it to be running on a specific node when doing a maintenance on a standby node, for example, when upgrading packages.

To stop the cluster, use the following command from one of the cluster nodes:

```
[root@onmssrv01 ~]# pcs cluster stop --all
onmssrv02: Stopping Cluster (pacemaker)...
onmssrv01: Stopping Cluster (pacemaker)...
onmssrv02: Stopping Cluster (corosync)...
onmssrv01: Stopping Cluster (corosync)...
```

To start the cluster, use the following command from one of the cluster nodes:

```
[root@onmssrv01 ~]# pcs cluster start --all
onmssrv01: Starting Cluster (pacemaker)...
onmssrv02: Starting Cluster (pacemaker)...
onmssrv01: Starting Cluster (corosync)...
onmssrv02: Starting Cluster (corosync)...
```

To stop the cluster services on a given node:

```
[root@onmssrv01 ~]# pcs cluster stop onmssrv02
onmssrv02: Stopping Cluster (pacemaker)...
onmssrv02: Stopping Cluster (corosync)...
```

To start the cluster services on a given node:

```
[root@onmssrv01 ~]# pcs cluster start onmssrv02
onmssrv02: Starting Cluster (pacemaker)...
onmssrv02: Starting Cluster (corosync)...
```

Pending

If you remember how the cluster was configured with *CMAN/Rgmanager*, you can infer that there is one feature that was not implemented with *Pacemaker*.

With *rgmanager*, we've created a failover domain with a `nofailback` property. That means, when the primary node dies, the resources will be moved to the secondary node, but they will remain running on that node even if the primary is back to business.

On the other hand, this is not the case with the resources on *Pacemaker*. There is a mandatory constraint created automatically when the resource group is created that gives priority to the node on which the group was created, in this case *onmssrv01*. That means, if *onmssrv01* has a problem, when it is back online, all the resources will be moved to this server.

I haven’t found a way to behave like *Rgmanager*.