

Node 1: 192.168.1.76

Node2:192.168.1.68

Machines: 2 Centos 6.5 can be accessible at both the ends

Add these nodes in hosts file on both nodes

vi /etc/hosts

192.168.1.76 node1

192.168.1.68 node2

1. Set up NTP and DNS for both your Linux Cluster nodes.

2. Add repository

Add HA clustering repo from Centos6.5 on both nodes! You will need this Repository to install CRM SHELL to manage Pacemaker resources:

***vi /etc/yum.repos.d/ha-clustering.repo***

[haclustering]

name=HA Clustering

baseurl=http://download.opensuse.org/repositories/network:/ha-clustering:/Stable/CentOS\_CentOS-6/

enabled=1

gpgcheck=0

3. Install packages

Install Corosync, Pacemaker and CRM Shell. Run this command on both Linux Cluster nodes:

**yum install pacemaker corosync crmsh -y**

4. Create configuration

Create Corosync configuration file which must be located in *“/etc/corosync/”* folder. You can copy /paste the following configuration to the IP address of your first Linux Cluster node:

***vi /etc/corosync/corosync.conf***



Copy Corosync configuration file to the second Linux Cluster node2 and add bindaddr: node2

5. Generate Auth Key

Generate Corosync Authentication Key by running *“corosync-keygen” – This might take some time!.*The key is located in *“/etc/corosync”* directory, file is named *“authkey”:*

[root@node1 /]# **corosync-keygen**

Corosync Cluster Engine Authentication key generator.

Gathering 1024 bits for key from /dev/random.

Press keys on your keyboard to generate entropy.

Press keys on your keyboard to generate entropy (bits = 176).

Press keys on your keyboard to generate entropy (bits = 240).

Press keys on your keyboard to generate entropy (bits = 304).

Press keys on your keyboard to generate entropy (bits = 368).

Press keys on your keyboard to generate entropy (bits = 432).

Press keys on your keyboard to generate entropy (bits = 496).

Press keys on your keyboard to generate entropy (bits = 560).

Press keys on your keyboard to generate entropy (bits = 624).

Press keys on your keyboard to generate entropy (bits = 688).

Press keys on your keyboard to generate entropy (bits = 752).

Press keys on your keyboard to generate entropy (bits = 816).

Press keys on your keyboard to generate entropy (bits = 880).

Press keys on your keyboard to generate entropy (bits = 944).

Press keys on your keyboard to generate entropy (bits = 1008).

Writing corosync key to /etc/corosync/authkey.

Transfer the *“/etc/corosync/authkey”* file to the second Linux Cluster node.

6. Start Corosync service on both nodes:

[root@node1 /]# **service corosync start**

Starting Corosync Cluster Engine (corosync):               [  OK  ]

[root@node1 /]# **service corosync start**

Starting Corosync Cluster Engine (corosync):               [  OK  ]

7. Start Pacemaker service on both nodes:

[root@node2 /]# **service pacemaker start**

Starting Pacemaker Cluster Manager:                        [  OK  ]

[root@node2 ~]# **service pacemaker start**

Starting Pacemaker Cluster Manager:                        [  OK  ]

8. Check cluster status

After a few seconds you can check your Linux Cluster status with *“crm status”* command:

[root@node1 /]# **crm status**

Last updated: Thu Sep 19 15:28:49 2013

Last change: Thu Sep 19 15:11:57 2013 via crmd on node1

Stack: classic openais (with plugin)

Current DC: node1 - partition with quorum

Version: 1.1.9-2.2-2db99f1

2 Nodes configured, 2 expected votes

0 Resources configured.

Online: [ node1 node2 ]

As we can see the status says 2 nodes are configured in this Linux Cluster – node1 and node2 Both nodes are online. Current DC is node1

NEXT STEP is to configure Pacemaker resources – applications, IP addresses in the cluster.

[root@node1 ~]# **crm help**

### View Linux Cluster Configuration

[root@node11 ~]# **crm configure show**

node node1

node node2

property $id="cib-bootstrap-options"

    dc-version="1.1.9-2.6-2db99f1"

    cluster-infrastructure="classic openais (with plugin)"

    expected-quorum-votes="2"

**Before we start adding Resources to our Cluster we need to disable STONITH**(Shoot The Other Node In The Head) – since we are not using it in our configuration**:**

## [root@node1 ~]# crm configure property stonith-enabled=false

If you in a 2 node cluster stops one of the two nodes, the node which is up fails, because the voting system fails.

So disable QUORUM

[root@node1 ~]#  **crm configure property no-quorum-policy=ignore**

### Adding Floating IP Address Resource

Let’s add **IP address**resource to our Linux Cluster. The information we need to configure IP address is:

**Cluster Resource Name:** CLUSTERIP  
**Resource Agent:** ocf:heartbeat:IPaddr2 (get this info with “crm ra meta IPaddr2”)  
**IP address:** 192.168.1.150  
**Netmask:** 24  
**Monitor interval:** 30 seconds (get this info with “crm ra meta IPaddr2”)

**Run the following command** on a Linux Cluster node to configure ClusterIP resource:

[root@node1 ~]# **crm configure primitive  CLUSTERIP ocf:heartbeat:IPaddr2 params ip=192.168.1.150 cidr\_netmask="24" op monitor interval="30s**"

**Check Cluster Configuration** with:

[root@node1 ~]# **crm configure show**

node node1

node node2

primitive  CLUSTERIP ocf:heartbeat:IPaddr2

    params ip="192.168.61.150" cidr\_netmask="24"

    op monitor interval="30s"

property $id="cib-bootstrap-options"

    dc-version="1.1.9-2.6-2db99f1"

    cluster-infrastructure="classic openais (with plugin)"

    expected-quorum-votes="2"

    stonith-enabled="false"

    last-lrm-refresh="1381240623"

[root@node1 ~]# **crm status**

Last updated: Tue Oct  8 15:59:19 2013

Last change: Tue Oct  8 15:58:11 2013 via cibadmin on node1

Stack: classic openais (with plugin)

Current DC: node1 - partition with quorum

Version: 1.1.9-2.6-2db99f1

2 Nodes configured, 2 expected votes

1 Resources configured.

Online: [node1 node2]

 CLUSTERIP   (ocf::heartbeat:IPaddr2):    Started node1

As we can see a new resource called CLUSTERIP **is configured in the Cluster and started on node1**

### Adding Apache (httpd) Resource

**yum install httpd** on both the servers

Next resource is an **Apache Web Server.** Prior to Apache Cluster Resource Configuration, **httpd package must be installed** and configured on **both nodes!** The information we need to configure Apache Web Server is:

**Cluster Resource Name:** Apache  
**Resource Agent:** ocf:heartbeat:apache (get this info with “crm ra meta apache”)  
**Configuration file location:** /etc/httpd/conf/httpd.conf  
**Monitor interval:** 30 seconds (get this info with “crm ra meta apache”)  
**Start timeout:**40 seconds (get this info with “crm ra meta apache”)  
**Stop timeout:**60 seconds (get this info with “crm ra meta apache”)

**Run the following command** on a Linux Cluster node to configure Apache resource:

[root@node1 ~]# **crm configure primitive Apache ocf:heartbeat:apache params configfile=/etc/httpd/conf/httpd.conf op monitor interval="30s" op start timeout="40s" op stop timeout="60s"**

**Check Cluster Configuration** with:

[root@node1 ~]# **crm configure show**

node node1

node node2

primitive Apache ocf:heartbeat:apache

    params configfile="/etc/httpd/conf/httpd.conf"

    op monitor interval="30s"

    op start timeout="40s" interval="0"

    op stop timeout="60s" interval="0"

    meta target-role="Started"

primitive CLUSTERIP ocf:heartbeat:IPaddr2

    params ip="192.168.61.150" cidr\_netmask="24"

    op monitor interval="30s"

property $id="cib-bootstrap-options"

    dc-version="1.1.9-2.6-2db99f1"

    cluster-infrastructure="classic openais (with plugin)"

    expected-quorum-votes="2"

    stonith-enabled="false"

    last-lrm-refresh="1381240623"

**Check Cluster Status** with:

[root@node1 ~]# **crm status**

Last updated: Thu Oct 10 11:13:59 2013

Last change: Thu Oct 10 11:07:38 2013 via cibadmin on node1

Stack: classic openais (with plugin)

Current DC: node1 - partition with quorum

Version: 1.1.9-2.6-2db99f1

2 Nodes configured, 2 expected votes

2 Resources configured.

Online: [ node1 node2 ]

 ClusterIP    (ocf::heartbeat:IPaddr2):    Started node1

 Apache    (ocf::heartbeat:apache):    Started node2

As we can see **both** **Cluster Resources (Apache and ClusterIP) are configured and started** – ClusterIP is started on node1. Cluster node and Apache is started on node2 node.

Apache and ClusterIP are at the moment running on different Cluster nodes but we will fix this later,  setting Resource Constraints like: colocation (colocating resources), order (order in which resources start and stop), …

# Deleting Cluster Resources

We can **delete** the configured Cluster Resources with “crm configure delete” command following by aResource Name we want to delete (example:)

**crm configure delete resourcename**

**We must always stop the Cluster Resource prior to deleting it!!**

We can stop the Resource by running “crm resource stop” command following by a Resource Name we want to stop.

We can **check the Linux Cluster configuration** by running “crm configure show” command and see, if the Cluster Resource was successfully removed from Cluster Configuration.

### Deleting Apache (httpd) Resource

Let’s**stop and delete** our **Apache** Cluster Resource configured in the steps above:

[root@node1 ~]# **crm resource stop Apache**

[root@node1 ~]# **crm configure delete Apache**

**Check Cluster Configuration** with:

[root@node1 ~]# **crm configure show**

node node1

node node2

primitive CLUSTERIP ocf:heartbeat:IPaddr2

    params ip="192.168.61.150" cidr\_netmask="24"

    op monitor interval="30s"

property $id="cib-bootstrap-options"

    dc-version="1.1.9-2.6-2db99f1"

    cluster-infrastructure="classic openais (with plugin)"

    expected-quorum-votes="2"

    stonith-enabled="false"

    last-lrm-refresh="1381240623"

… to confirm **Apache** resource was **deleted** from Cluster Configuration.

### Deleting IP Address Resource

Next let’s **stop and delete** **ClusterIP** Resource:

[root@node1 ~]# **crm resource stop ClusterIP**

[root@node1 ~]# **crm configure delete ClusterIP**

**Check Cluster Configuration** with:

[root@node1 ~]# **crm configure show**

node node1

node node2

property $id="cib-bootstrap-options"

    dc-version="1.1.9-2.6-2db99f1"

    cluster-infrastructure="classic openais (with plugin)"

    expected-quorum-votes="2"

    stonith-enabled="false"

    last-lrm-refresh="1381240623"

… to confirm the **ClusterIP** Resource was **deleted** from our Cluster Configuration.

# Cluster Resource Management

[CRM Shell](http://savannah.nongnu.org/projects/crmsh/) is used for Linux Cluster management. We can use “crm configure” with “group, order, location, colocation, …” parameters and “crm resource” with “start, stop, status, migrate, cleanup, …“.

The following examples cover the **basic Linux Cluster resource management commands you might find useful**. Additional help is available by executing “crm configure help” or “crm resource help” command.

### ****Our current Linux Cluster resource configuration is:****

[root@node1 ~]# **crm configure show**

node node1

node node2

primitive Apache ocf:heartbeat:apache

    params configfile="/etc/httpd/conf/httpd.conf"

    op monitor interval="30s"

    op start timeout="40s" interval="0"

    op stop timeout="60s" interval="0"

primitive ClusterIP ocf:heartbeat:IPaddr2

    params ip="192.168.1.150" cidr\_netmask="24"

    op monitor interval="30s"

property $id="cib-bootstrap-options"

    dc-version="1.1.10-1.el6\_4.4-368c726"

    cluster-infrastructure="classic openais (with plugin)"

    expected-quorum-votes="2"

    stonith-enabled="false"

    last-lrm-refresh="1383902488"

### Group Linux Cluster Resources “crm configure group groupname resource1 resource2″

Group your Linux Cluster resources and start/stop and manage your resource group with one single command.

[root@node1 ~]# **crm configure group HTTP-GROUP ClusterIP Apache**

[root@node1 ~]# **crm configure show**

node node1

node node2

primitive Apache ocf:heartbeat:apache

params configfile="/etc/httpd/conf/httpd.conf"

op monitor interval="30s"

op start timeout="40s" interval="0"

op stop timeout="60s" interval="0"

primitive ClusterIP ocf:heartbeat:IPaddr2

params ip="192.168.1.150" cidr\_netmask="24"

op monitor interval="30s"

group HTTP-GROUP ClusterIP Apache

property $id="cib-bootstrap-options"

dc-version="1.1.10-1.el6\_4.4-368c726"

cluster-infrastructure="classic openais (with plugin)"

expected-quorum-votes="2"

stonith-enabled="false"

last-lrm-refresh="1383902488"

In this example we created a resource group called HTTP-GROUP with ClusterIP and Apache resources. We can now manage all our grouped resources by starting, stopping and managing HTTP-GROUP group resource.

### Linux Cluster Resources Start/Stop Order “crm configure order ordername inf: resource1 resource2:start“

With this command we can configure start and stop order of our Linux Cluster resources.

[root@node1 ~]# **crm configure order ClusterIP-before-Apache inf: ClusterIP Apache:start**

[root@node1 ~]# **crm configure show**

node node1

node node2

primitive Apache ocf:heartbeat:apache

    params configfile="/etc/httpd/conf/httpd.conf"

    op monitor interval="30s"

    op start timeout="40s" interval="0"

    op stop timeout="60s" interval="0"

primitive ClusterIP ocf:heartbeat:IPaddr2

    params ip="192.168.1.150" cidr\_netmask="24"

    op monitor interval="30s"

order ClusterIP-before-Apache inf: ClusterIP Apache:start

property $id="cib-bootstrap-options"

    dc-version="1.1.10-1.el6\_4.4-368c726"

    cluster-infrastructure="classic openais (with plugin)"

    expected-quorum-votes="2"

    stonith-enabled="false"

    last-lrm-refresh="1383902488"

In this example we configured the start and stop order of our ClusterIP and Apache resources. As configured, ClusterIP resource will start first and only then Apache resource can be started. When stopping, Apache resource will be stopped and only then ClusterIP resource can be stopped too.

### Linux Cluster Resources Colocation “crm configure colocation colocationname inf: resource1 resource2“

We can configure Linux Cluster resources colocation. Like said we colocate the desired resources and make sure we always run desired resources on the same node at all time.

[root@node1 ~]# **crm configure colocation IP-with-APACHE inf: ClusterIP Apache**

[root@node1 ~]# **crm configure show**

node node2

node node2

primitive Apache ocf:heartbeat:apache

    params configfile="/etc/httpd/conf/httpd.conf"

    op monitor interval="30s"

    op start timeout="40s" interval="0"

    op stop timeout="60s" interval="0"

primitive ClusterIP ocf:heartbeat:IPaddr2

    params ip="192.168.1.150" cidr\_netmask="24"

    op monitor interval="30s"

group HTTP-GROUP ClusterIP Apache

colocation IP-with-APACHE inf: ClusterIP Apache

order ClusterIP-before-Apache inf: ClusterIP Apache:start

property $id="cib-bootstrap-options"

    dc-version="1.1.10-1.el6\_4.4-368c726"

    cluster-infrastructure="classic openais (with plugin)"

    expected-quorum-votes="2"

    stonith-enabled="false"

    last-lrm-refresh="1384349363"

In this example, we configured colocation for ClusterIP and Apache resources. ClusterIP and Apache resources will always be started and running together, on the same Linux Cluster node.

### Linux Cluster Resources Prefered Location “crm configure location locationname resource score: clusternode“

We can configure a prefered location for our Linux Cluster resources or resource groups. We must always set the location score – ositive values indicate the resource should run on this node. Negative values indicate the resource should not run on this node.

[root@node1 ~]# **crm configure location HTTP-GROUP-prefer-NODE1 HTTP-GROUP 50: node1**

[root@node1 ~]# **crm configure show**

node node1

node node2

primitive Apache ocf:heartbeat:apache

    params configfile="/etc/httpd/conf/httpd.conf"

    op monitor interval="30s"

    op start timeout="40s" interval="0"

    op stop timeout="60s" interval="0"

primitive ClusterIP ocf:heartbeat:IPaddr2

    params ip="192.168.61.150" cidr\_netmask="24"

    op monitor interval="30s"

group HTTP-GROUP ClusterIP Apache

location HTTP-GROUP-prefer-NODE1 HTTP-GROUP 50: node1

colocation IP-with-APACHE inf: ClusterIP Apache

order ClusterIP-before-Apache inf: ClusterIP Apache:start

property $id="cib-bootstrap-options"

    dc-version="1.1.10-1.el6\_4.4-368c726"

    cluster-infrastructure="classic openais (with plugin)"

    expected-quorum-votes="2"

    stonith-enabled="false"

    last-lrm-refresh="1384349363"

In this example we configured the prefered location of HTTP-GROUP resource group. By configuring score 50, HTTP-GROUP will prefer to run on node1 node but will still in case of node1 failure move to node2. When node1 recovers, HTTP-GROUP will move back to prefered node1.

### ****Checking the status of our Linux Cluster Nodes and Resources:****

[root@node1 ~]# **crm status**

Last updated: Wed Nov 13 15:30:45 2013

Last change: Wed Nov 13 15:01:06 2013 via cibadmin on node1

Stack: classic openais (with plugin)

Current DC: node2 - partition with quorum

Version: 1.1.10-1.el6\_4.4-368c726

2 Nodes configured, 2 expected votes

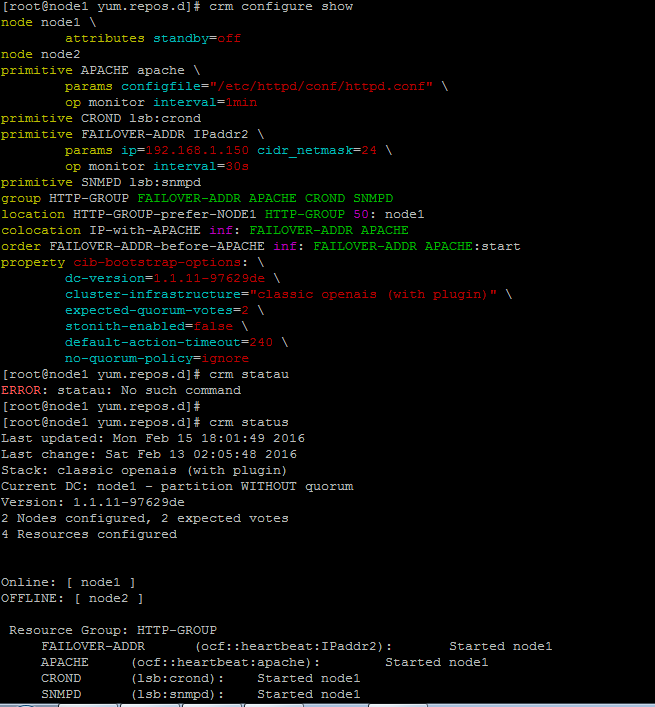
2 Resources configured

Online: [ node1 node2 ]

 Resource Group: HTTP-GROUP

     ClusterIP    (ocf::heartbeat:IPaddr2):    Started node1

     Apache    (ocf::heartbeat:apache):    Started node1



**Adding LSB Agents**

## [root@node1 ~]# crm configure primitive anydeamon lsb:anydeamon op monitor interval="10s"

## [root@node1 ~]# crm configure primitive CROND lsb:cron op monitor interval="10s"

## [root@node1 ~]# crm configure primitive SNMPD lsb:snmpd op monitor interval="10s"

## Init Script (LSB) Compatibility Checks

Assuming *some\_service* is configured correctly and currently not active, the following sequence will help you determine if it is LSB compatible:

1. Start (stopped)

/etc/init.d/some\_service start ; echo "result: $?"

* + Did the service start?
  + Did the command print result: 0 (in addition to the regular output)?

1. Status (running)

/etc/init.d/some\_service status ; echo "result: $?"

* + Did the script accept the command?
  + Did the script indicate the service was running?
  + Did the command print result: 0 (in addition to the regular output)?

1. Start (running)

/etc/init.d/some\_service start ; echo "result: $?"

* + Is the service still running?
  + Did the command print result: 0 (in addition to the regular output)?

1. Stop (running)

/etc/init.d/some\_service stop ; echo "result: $?"

* + Was the service stopped?
  + Did the command print result: 0 (in addition to the regular output)?

1. Status (stopped)

/etc/init.d/some\_service status ; echo "result: $?"

* + Did the script accept the command?
  + Did the script indicate the service was **not** running?
  + Did the command print result: 3 (in addition to the regular output)?

1. Stop (stopped)

/etc/init.d/some\_service stop ; echo "result: $?"

* + Is the service still stopped?
  + Did the command print result: 0 (in addition to the regular output)?

1. Status (failed)

This step is not readily testable and relies on manual inspection of the script.

The script can optionally use one of the other codes (other than 3) listed in the LSB spec to indicate that it is active but failed.

In such a case, this tells the cluster that, before moving the resource to another node, it should stop it on the existing one first.

Making use of these extra exit codes is encouraged.

If the answer to any of the above questions is ***no***, then the init script is not LSB compliant.

If you are using **Pacemaker** resource management, then your options at this point are to: