**How to configure GPDB and GPCC High Availability with pacemaker**

Apr, 2023

VMware Tanzuu Support

Staff Product Support Engineer

Jack Moon

**[ High Availability Clustering Explanation]**

For those not familiar with High Availability resources & services or Cluster Computing, in this guide a brief summary and explanation of OCF Resource Agents will be reviewed before diving into creating own for GPCC. In cluster networking multiple servers are inter-connected together on a network to provide improved delivery of services such as with HA.

The servers ( referred to as nodes ) are all managed by a Cluster Resource Manager ( CRM ) which manages cluster resources. High Availability (HA) refers to the ability to provide a resource or service with minimal to zero downtime. Whatever client wants, client gets, and the store is open 24/7/365. In this case Pacemaker will be using as CRM.

Now each node has to have a way to communicate with other nodes in cluster for this to work, when a resource fails, or a node goes down, the CRM needs to be notified to make other resources available. This is where Corosync steps in. Corosync has the communication layer that provides Pacemaker with the status of resources and nodes on the cluster. The information that Pacemaker receives can communicate many complex situations and scenarios that Pacemaker can deal with in a variety of ways.

**[ OCF Resource Agents ]**

Where do the Resource Agents fit into this step? the RAs are deployed by pacemaker to interface with each resource that needs managed, using Corosync’s communication layer. The protocol used by the RAs is known as the Open Cluster Framework (OCF) and defines how RAs should be implemented and requirements to create one. Most of the time they are implemented in a shell script, but they can be written in any programming language needed. More info on HA cluster computing and OCF protocol implementation can be found at [Cluster Labs](http://clusterlabs.org/).

**[ Environment ]**

1) OS : RHEL or Rocky Linux 8 ( RHEL, CentOS, Rocky Linux 7.x version would be possible )

2) GPDB 6.22.x and GPCC 6.8.x

3) A master and standby host, 3 segment hosts

* Master : rk8-master , 192.168.0.171
* Standby : rk8-slave , 192.168.0.172
* Segments : rk8-node0[1-3] , 192.168.0.173 ~ 175

**[ Prerequisites ]**

This guide assumes that GPDB cluster is already deployed and GPCC set up on master and standby host. Accessing the remote nodes via ssh / scp should transfer the completed files. Note that every node will need a copy of the Resource Agent script.

[1] rk8-master-vip entry should be included in /etc/hosts as it’s virtual ip of gpdb service

| [root@rk8-master ~]# cat /etc/hosts  127.0.0.1 localhost localhost.localdomain localhost4 localhost4.localdomain4  ::1 localhost localhost.localdomain localhost6 localhost6.localdomain6  192.168.0.171 rk8-master.jtest.pivotal.io rk8-master  192.168.0.172 rk8-slave.jtest.pivotal.io rk8-slave  ~~ snip  **192.168.0.180 rk8-master-vip.jtest.pivotal.io rk8-master-vip** |
| --- |

[2] gpcc will connect to virtual ip of gpdb service so that it could be worked in case of failover.

| [root@rk8-master ~]# cat /usr/local/greenplum-cc/conf/app.conf  appname = gpccws  listentcp4 = true  runmode = prod  session = true  ~~ snip  master\_port = 5432  **master\_host = rk8-master-vip**  path = /usr/local  display\_name = gpcc  ~~ snip  [root@rk8-slave ~]# cat /usr/local/greenplum-cc/conf/app.conf  appname = gpccws  listentcp4 = true  runmode = prod  session = true  ~~ snip  master\_port = 5432  **master\_host = rk8-master-vip**  path = /usr/local  display\_name = gpcc  ~~ snip |
| --- |

**[ Design ]**

Creating a simple Resource Agent that runs a supplied script perform a health check of gpdb and gpcc. gpcc\_monitor and gpdb\_monitory will simply check if the GPCC and GPDB is running now.

If the gpcc\_monitor does not pass, then gpccws should be restarted on same node a few times and then migrate to another node if restarting is failed. If the gpdb\_monitor does not pass, then gpactivatestandby will immediately run to become master role on the standby host and virtual-ip address will take over into standby host as well.

**[ Installation and configuration ]**

**[1] Disable Firewall and SELinux on master and standby host both and Reboot.**

| $ systemctl stop firewalld $ systemctl disable firewalld  $ vi /etc/sysconfig/selinux  # This file controls the state of SELinux on the system.  # SELINUX= can take one of these three values:  # enforcing - SELinux security policy is enforced.  # permissive - SELinux prints warnings instead of enforcing.  # disabled - No SELinux policy is loaded.  SELINUX=disabled  # SELINUXTYPE= can take one of these three values:  # targeted - Targeted processes are protected,  # minimum - Modification of targeted policy. Only selected processes are protected.  # mls - Multi Level Security protection.  SELINUXTYPE=targeted  $ reboot |
| --- |

**[2] Install the pacemaker packages with the components on both master and standby after above firewall and selinux set up as basic.**

| # Enable High Availability Repository  $ yum-config-manager --enable ha # or dnf config-manager --set-enabled ha  $ yum -y install corosync pacemaker pcs fence-agents-all  $ systemctl enable pcsd corosync pacemaker |
| --- |

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

| $ echo "changeme" | passwd --stdin hacluster  passwd: all authentication tokens updated successfully. |
| --- |

**[4] Next, start and enable pcsd service on both nodes.**

| $ systemctl start pcsd  $ systemctl enable pcsd |
| --- |

**[5] Since all nodes will be configured from one point, authentication is needed on all nodes before the configuration is allowable to change. Use the previously configured hacluster user and password to do this.**

| $ pcs host auth rk8-master rk8-slave -u hacluster -p changeme  Username: hacluster  Password:  rk8-master: Authorized  rk8-slave: Authorized |
| --- |

From here, using PCS can control the cluster from rk8-master. It’s no longer required to repeat all commands on both nodes.

**[6] Create the cluster and add nodes. Adding/Setting both nodes starts cluster named gpcc-cluster.**

| $ pcs cluster setup gpcc-cluster rk8-master rk8-slave  ...  rk8-master: Succeeded  rk8-slave: 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.

**[7] After creating the cluster and adding nodes to it, it can be started. The cluster won’t do a lot yet since any resources have not been configured.**

| $ pcs cluster start --all  rk8-master: Starting Cluster...  rk8-slave: Starting Cluster... |
| --- |

The pacemaker and corosync services are started on both nodes (as will happen at boot time) to accomplish this.

**[8] To check the status of the cluster after starting it.**

| $ pcs status cluster  Cluster Status:  Status of pacemakerd: 'Pacemaker is running' (last updated 2023-02-16 21:19:42 +09:00)  Cluster Summary:  \* Stack: corosync  \* Current DC: rk8-slave (version 2.1.4-5.el8\_7.2-dc6eb4362e) - partition with quorum  \* Last updated: Thu Feb 16 21:19:42 2023  \* Last change: Thu Feb 16 11:46:32 2023 by root via crm\_resource on rk8-slave  \* 2 nodes configured  \* 0 resource instances configured  Node List:  \* Online: [ rk8-master rk8-slave ]  PCSD Status:  rk8-master: Online  rk8-slave: Online |
| --- |

**[9] To check the status of the nodes in the cluster.**

| $ pcs status nodes  Pacemaker Nodes:  Online: rk8-master rk8-slave  Standby:  Standby with resource(s) running:  Maintenance:  Offline:  Pacemaker Remote Nodes:  Online:  Standby:  Standby with resource(s) running:  Maintenance:  Offline:  $ corosync-cmapctl | grep members  runtime.members.1.config\_version (u64) = 0  runtime.members.1.ip (str) = r(0) ip(192.168.0.171)  runtime.members.1.join\_count (u32) = 1  runtime.members.1.status (str) = joined  runtime.members.2.config\_version (u64) = 0  runtime.members.2.ip (str) = r(0) ip(192.168.0.172)  runtime.members.2.join\_count (u32) = 1  runtime.members.2.status (str) = joined  $ pcs status corosync  Membership information  ----------------------  Nodeid Votes Name  1 1 rk8-master (local)  2 1 rk8-slave |
| --- |

**[10] Cluster configuration**

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

| $ 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 is still an error regarding STONITH ( Shoot The Other Node In The Head ), 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.

Above error will be disappeared after all resources are configured correctly.

**[11] Since two node cluster will be deployed, the stonith option would not be enable.**

| $ pcs property set stonith-enabled=false |
| --- |

**[12] Configure the quorum settings to ignore a low quorum while configuring the behavior of the cluster.**

| $ pcs property set no-quorum-policy=ignore  $ pcs property  Cluster Properties:  cluster-infrastructure: corosync  cluster-name: gpcc-cluster  dc-version: 2.1.4-5.el8\_7.2-dc6eb4362e  have-watchdog: false  no-quorum-policy: ignore  stonith-enabled: false |
| --- |

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 two 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.

**[13] A virtual IP will be added to cluster. This virtual IP is the IP address which will be contacted to reach the services ( the GPDB in this case ). A virtual IP is a resource. To add the resource:**

| $ pcs resource create gpdb-vip ocf:heartbeat:IPaddr2 ip=192.168.0.180 cidr\_netmask=24 nic=eth0 iflabel=0 op monitor interval=10s on-fail=standby  gpcc-vip (ocf::heartbeat:IPaddr2): Started |
| --- |

**[14] Looking at the above command, the resource is marked as started. So the new, virtual, IP address should be reachable.**

| $ ping -c1 192.168.0.180  PING 192.168.0.180 (192.168.0.180) 56(84) bytes of data.  64 bytes from 192.168.0.179: icmp\_seq=1 ttl=64 time=0.164 ms  --- 192.168.0.179 ping statistics ---  1 packets transmitted, 1 received, 0% packet loss, time 0ms  rtt min/avg/max/mdev = 0.164/0.164/0.164/0.000 ms |
| --- |

**[15] To see who is the current owner of the Resource/Virtual IP:**

| $ pcs status | grep gpdb-vip  \* gpdb-vip (ocf::heartbeat:IPaddr2): Started rk8-master |
| --- |

**[16] Start GPDB in cluster**Now GPDB will be started on master node. From now on, the cluster will be responsible for monitoring if gpdb is running on the master and becoming master role on standby host when master fails. First start gpdb:

| $ su - gpadmin $ source $GPHOME/greenplum\_path.sh && gpstart -a |
| --- |

Now that GPDB is running as master role on master host and a resource will be added for the Cluster. Remember that we only need to do this from one node since twol nodes are configured by PCS.

**[17] A template OCF script need to make a copy and rename the RA something pertinent like “gpdb” from “Dummy”**

| $ cp /usr/lib/ocf/resource.d/heartbeat/Dummy /usr/lib/ocf/resource.d/heartbeat/gpdb |
| --- |

**[18] Ensure to make the copy executable**

| $ chmod +x /usr/lib/ocf/resource.d/heartbeat/gpdb |
| --- |

**[19] Change names in the gpdb script to the one we renamed it to**

| $ sed -i 's/Dummy/gpdb/g' /usr/lib/ocf/resource.d/heartbeat/gpdb  $ sed -i 's/dummy/gpdb/g' /usr/lib/ocf/resource.d/heartbeat/gpdb |
| --- |

**[20] Add actions to start, stop and status in RA of GPDB.**

| ~~ snip  gpdb\_start() {  **if [ -z "$(ps -ef | grep postgres | grep 'wal receiver' | grep -v grep)" ] && [ -z "$(pidof postgres)" ]; then**  **su - gpadmin -c "source /usr/local/greenplum-db/greenplum\_path.sh && timeout -s 9 20s gpstart -a"**  **if [ $? = $OCF\_SUCCESS ]; then**  **touch ${OCF\_RESKEY\_state}**  **return $OCF\_SUCCESS # - OK**  **else**  **return $OCF\_ERR\_GENERIC**  **fi**  **else**  **su - gpadmin -c "source /usr/local/greenplum-db/greenplum\_path.sh && export PGPORT=5432 && gpactivatestandby -f -a -d /data/master/gpseg-1"**  **if [ $? = $OCF\_SUCCESS ]; then**  **touch ${OCF\_RESKEY\_state}**  **return $OCF\_SUCCESS # - OK**  **else**  **return $OCF\_ERR\_GENERIC**  **fi**  **fi**  }  gpdb\_stop() {  **if [ -z "$(ps -ef | grep postgres | grep 'wal receiver' | grep -v grep)" ]; then**  **su - gpadmin -c "source /usr/local/greenplum-db/greenplum\_path.sh && timeout -s 9 10s gpstop -a"**  **if [ $? = $OCF\_SUCCESS ]; then**  **rm ${OCF\_RESKEY\_state}**  **return $OCF\_SUCCESS**  **else**  **if [ ! -z "$(pidof postgres)" ]; then**  **kill -9 $(pidof postgres)**  **rm -rf /tmp/.s.PGSQL.\***  **rm ${OCF\_RESKEY\_state}**  **return $OCF\_SUCCESS**  **else**  **return $OCF\_SUCCESS**  **fi**  **fi**  **fi**  }  gpdb\_monitor() {  # Monitor \_MUST!\_ differentiate correctly between running  # (SUCCESS), failed (ERROR) or \_cleanly\_ stopped (NOT RUNNING).  # That is THREE states, not just yes/no.  **if [ -f ${OCF\_RESKEY\_state} ]; then**  **su - gpadmin -c "source /usr/local/greenplum-db/greenplum\_path.sh && timeout -s 9 5s gpstate -Q > /dev/null 2>&1"**  **if [ $? = $OCF\_SUCCESS ]; then**  **return $OCF\_SUCCESS**  **else**  **su - gpadmin -c "timeout -s 9 5s psql -h localhost -c \"select 100 from pg\_database where datname='gpadmin'\"" \**  **> /dev/null 2>&1**  **if [ $? = $OCF\_SUCCESS ]; then**  **return $OCF\_SUCCESS**  **else**  **if [ -z "$(ps -ef | grep postgres | grep 'wal sender')" ]; then**  **kill -9 $(pidof postgres)**  **rm -rf /tmp/.s.PGSQL.\***  **rm ${OCF\_RESKEY\_state}**  **else**  **kill -9 $(pidof postgres)**  **rm -rf /tmp/.s.PGSQL.\***  **rm ${OCF\_RESKEY\_state}**  **fi**  **fi**  **fi**  **fi**  if false ; then  return $OCF\_ERR\_GENERIC  fi  if ! ocf\_is\_probe && [ "$\_\_OCF\_ACTION" = "monitor" ]; then  # set exit string only when NOT\_RUNNING occurs during an actual monitor operation.  ocf\_exit\_reason "No process state file found"  fi  return $OCF\_NOT\_RUNNING  }  ~~ snip |
| --- |

Now a Resource Agent can be run and managed with a shell script. Arguments will be passed through the agent to the calling script which is supplied by it.

**[21] Now copy it over to each of the cluster nodes, install and test it.**

| $ scp /usr/lib/ocf/resource.d/heartbeat/gpdb root@rk8-slave:/usr/lib/ocf/resource.d/heartbeat/ |
| --- |

**[22] Test your agent using ocf-tester to catch any mistakes (on remote nodes). We want to do this on all nodes to ensure they are configured / have correct dependencies to run our script. Node: provide it with full path to script.**

| $ ocf-tester -n /usr/lib/ocf/resource.d/heartbeat/gpdb |
| --- |

**[23] Try executing gpdb resource agent in a shell and make sure it executes w/o any major errors. Again like previous example we should check each of the nodes.**

| $ export OCF\_ROOT=/usr/lib/ocf; bash -x /usr/lib/ocf/resource.d/heartbeat/gpdb start # or stop, monitor |
| --- |

**[24] Add the gpdb monitoring resource agent to Pacemaker using pcs.  
This should be done on all nodes in cluster you want that resource agent to monitor on.**

| $ pcs resource create gpdb ocf:heartbeat:gpdb op monitor timeout=10s interval=10s on-fail=standby |
| --- |

**[25] To see status of cluster**

| $ pcs status |
| --- |

**[26] Which should show your newly added resource agent and some additional status information that can be helpful in debugging. To show detailed information about only the gpdb monitoring resource agent run the following:**

| $ pcs resource show gpdb |
| --- |

**[27] You can also run gpdb resource agent action with the same cmd. This can be useful to test if a certain action isn’t working properly.**

| $ pcs resource debug-start gpdb  $ pcs resource debug-stop gpdb  $ pcs resource debug-monitor gpdb |
| --- |

**[28] A template OCF script need to make a copy and rename the RA something pertinent like “gpcc” from “Dummy”**

| $ cp /usr/lib/ocf/resource.d/heartbeat/Dummy /usr/lib/ocf/resource.d/heartbeat/gpcc |
| --- |

**[29] Ensure to make the copy executable**

| $ chmod +x /usr/lib/ocf/resource.d/heartbeat/gpcc |
| --- |

**[30] Change names in the gpcc script to the one we renamed it to**

| $ sed -i 's/Dummy/gpcc/g' /usr/lib/ocf/resource.d/heartbeat/gpcc  $ sed -i 's/dummy/gpcc/g' /usr/lib/ocf/resource.d/heartbeat/gpcc |
| --- |

**[31] Modify actions to start, stop and status in RA of GPCC.**

| $ vi /usr/lib/ocf/resource.d/heartbeat/gpcc  ~~ snip  gpcc\_start() {  **su - gpadmin -c "source /usr/local/greenplum-db/greenplum\_path.sh && source /usr/local/greenplum-cc/gpcc\_path.sh && gpcc start"**  **if [ $? = $OCF\_SUCCESS ]; then**  **return $OCF\_SUCCESS**  **fi**  touch ${OCF\_RESKEY\_state}  }  gpcc\_stop() {  **su - gpadmin -c "source /usr/local/greenplum-db/greenplum\_path.sh && source /usr/local/greenplum-cc/gpcc\_path.sh && timeout -s 9 10s gpcc stop"**  **if [ $? != 0 ]; then**  **kill -9 $(ps -ef | grep -v grep | grep gpccws | awk '{print $2}')**  **return 0**  **fi**  if [ $? = $OCF\_SUCCESS ]; then  rm ${OCF\_RESKEY\_state}  fi  return $OCF\_SUCCESS  }  gpcc\_monitor() {  # Monitor \_MUST!\_ differentiate correctly between running  # (SUCCESS), failed (ERROR) or \_cleanly\_ stopped (NOT RUNNING).  # That is THREE states, not just yes/no.  **( su - gpadmin -c "source /usr/local/greenplum-db/greenplum\_path.sh && source /usr/local/greenplum-cc/gpcc\_path.sh && gpcc status | grep -E 'webserver: running' > /dev/null 2>&1" ) && \**  **( su - gpadmin -c "source /usr/local/greenplum-db/greenplum\_path.sh && source /usr/local/greenplum-cc/gpcc\_path.sh && gpcc status | grep -E 'agents running' > /dev/null 2>&1" ) && \**  **( curl -I --insecure --connect-timeout 5 https://localhost:28080 > /dev/null 2>&1 )**  **if [ $? = $OCF\_SUCCESS ]; then**  **return $OCF\_SUCCESS**  **fi**  if false ; then  return $OCF\_ERR\_GENERIC  fi  if ! ocf\_is\_probe && [ "$\_\_OCF\_ACTION" = "monitor" ]; then  # set exit string only when NOT\_RUNNING occurs during an actual monitor operation.  ocf\_exit\_reason "No process state file found"  fi  return $OCF\_NOT\_RUNNING  }  ~~ snip |
| --- |

Now a GPCC Resource Agent can be run and managed with a shell script. Arguments will be passed through the agent to the calling script which is supplied by it.

**[32] Now copy it over to each of the cluster nodes, install and test it.**

| $ scp /usr/lib/ocf/resource.d/heartbeat/gpcc root@rk8-slave:/usr/lib/ocf/resource.d/heartbeat/ |
| --- |

**[33] Test your gpcc resource agent using ocf-tester to catch any mistakes (on remote nodes). We want to do this on all nodes to ensure they are configured / have correct dependencies to run our script. Node: provide it with full path to script.**

| $ ocf-tester -n /usr/lib/ocf/resource.d/heartbeat/gpcc |
| --- |

**[34] Try executing gpcc resource agent in a shell and make sure it executes w/o any major errors. Again like previous example we should check each of the nodes.**

| $ export OCF\_ROOT=/usr/lib/ocf; bash -x /usr/lib/ocf/resource.d/heartbeat/gpcc start # or status, monitor |
| --- |

**[35] Add the resource to Pacemaker using pcs.  
This should be done on all nodes in cluster you want that resource agent to monitor on.**

| $ pcs resource create gpcc ocf:heartbeat:gpcc op monitor timeout=10s interval=10s on-fail=standby |
| --- |

**[36] To see status of cluster**

| $ pcs status |
| --- |

**[37] Which should show your newly added resource agent and some additional status information that can be helpful in debugging. To show detailed information about only the health-agent resource run the following:**

| $ pcs resource show gpcc |
| --- |

**[38] You can also run resource agent action cmds with the same cmd. This can be useful to test if a certain action isn’t working properly.**

| $ pcs resource debug-start gpcc  $ pcs resource debug-stop gpcc  $ pcs resource debug-monitor gpcc |
| --- |

**[39] You can also run resource agent action cmds with the same cmd. This can be useful to test if a certain action isn’t working properly.**

| $ pcs resource create gpcc ocf:heartbeat:gpcc op monitor timeout=10s interval=10s |
| --- |

**# on-fail options will not be added on gpcc resource agent since it does not need to run on master host only and can be run on either master or slave node.**

**[40] Configure resource agents at which host should be prefer to run**

| $ pcs constraint location gpdb prefers rk8-master=50  $ pcs constraint location gpdb-vip prefers rk8-master=50  $ pcs constraint location gpcc prefers rk8-slave=50 |
| --- |

**[41] Configure gpdb and gpdb-vip resources always run on same node.**

**By default, the cluster will try to balance the resources over the cluster. That means that the gpdb, which is a resource, will be started on a different node than the gpdb-vip. Starting to run gpactivatestandby on a node that isn’t the owner of the Virtual IP will cause it to fail gpcc since we configured gpcc to connect into the Virtual IP. In order to make sure that the Virtual IP and gpdb monitoring agent always stay together, we can add a constraint:**

| $ pcs constraint colocation add gpdb with gpdb-vip INFINITY  $ pcs constraint order gpdb then gpdb-vip |
| --- |

**[42] Pacemaker has the concept of resource stickiness, which controls how strongly a service prefers to stay running where it is. You may like to think of it as the "cost" of any downtime. By default, Pacemaker assumes there is zero cost associated with moving resources and will do so to achieve "optimal" resource placement. We can specify a different stickiness for every resource. In this case resources will not stay for the certain period of time and take over right away into other node if these are failed. Because FAILED will be marked on host when gpdb, gpdb-vip resource fails and these would not be rollback automatically in this case.**

| $ pcs resource meta gpdb resource-stickiness=0  $ pcs resource meta gpdb-vip resource-stickiness=0  $ pcs resource meta gpcc resource-stickiness=0 |
| --- |

FYI, In most circumstances, it is highly desirable to prevent healthy resources from being moved around the cluster. Moving resources almost always requires a period of downtime. For complex services such as databases, this period can be quite long.

**[43] To look at the configured constraints:**

| [root@rk8-master ~]# pcs constraint config  Location Constraints:  Resource: gpcc  Enabled on:  Node: rk8-slave (score:50)  Resource: gpdb  Enabled on:  Node: rk8-master (score:50)  Resource: gpdb-vip  Enabled on:  Node: rk8-master (score:50)  Ordering Constraints:  start gpdb then start gpdb-vip (kind:Mandatory)  Colocation Constraints:  gpdb with gpdb-vip (score:INFINITY)  Ticket Constraints: |
| --- |

**[44] After configuring the cluster with the correct constraints, restart it and check the status.**

| [root@rk8-master ~]# pcs status  Cluster name: gpcc-cluster  Status of pacemakerd: 'Pacemaker is running' (last updated 2023-03-15 21:34:29 -04:00)  Cluster Summary:  \* Stack: corosync  \* Current DC: rk8-master (version 2.1.4-5.el8\_7.2-dc6eb4362e) - partition with quorum  \* Last updated: Wed Mar 15 21:34:29 2023  \* Last change: Wed Mar 15 21:28:44 2023 by hacluster via crmd on rk8-master  \* 2 nodes configured  \* 3 resource instances configured  Node List:  \* Online: [ rk8-master rk8-slave ]  Full List of Resources:  \* gpdb-vip (ocf::heartbeat:IPaddr2): Started rk8-master  \* gpdb (ocf::heartbeat:gpdb): Started rk8-master  \* gpcc (ocf::heartbeat:gpcc): Started rk8-slave  Daemon Status:  corosync: active/disabled  pacemaker: active/disabled  pcsd: active/enabled |
| --- |

As you can see, the Virtual IP and the gpcc are both running on rk8-master. If all goes well, you should be able to reach the GPCC Web UI on the Virtual IP address (192.168.0.180).

**[45] If you want to test the failover, you can stop the cluster for rk8-node01 and see if the website is still available on the virtual IP:**

| $ pcs cluster stop rk8-master  rk8-master: Stopping Cluster (pacemaker)...  rk8-master: Stopping Cluster (corosync)... |
| --- |

**[46] Check if resource are taken over to slave node.**

| $ pcs status  Cluster name: gpcc-cluster  ~~ snip  Node List:  \* Online: [ rk8-slave ]  \* OFFLINE: [ rk8-master ]  Full List of Resources:  \* gpcc-vip (ocf::heartbeat:IPaddr2): Started rk8-slave  \* gpcc (ocf::heartbeat:gpcc): Started rk8-slave  ~~ snip |
| --- |

A refresh of the same URL gives the GPCC Web UI served by rk8-slave:28080. Since gpccws migrated to rk9-slave we could see Web UI there.

**[47]** Test failover resources to standby node

| **[A] Just kill gpccws process**  $ killall gpccws  # Check if gpccws is restarted or failovered  $ ps axf  **[B] Make gpccws to zombie process**  $ ps -ef | grep greenplum-cc ~~ snip  41935 ? Sl 0:00 /usr/local/greenplum-cc-6.8.4/bin/gpccws….  42097 ? Sl 0:00 /usr/local/greenplum-cc-6.8.4/bin/ccaagent….  $ gdb -p 41935  ~~ snip  (gdb)  # Open a new terminal and kill process of gpccws  $ kill -9 41935  # Check if process is now zombile  $ ps -ef | grep gpccws  ~~ snip  41935 ? Zl 0:00 [gpccws] <defunct>  # Check if process is now restarted or failover.  $ ps axf  **[C] Make unresponsive status of gpccws**  $ kill -STOP <pid> # -CONT will be resume.  $ Checking the status of gpccws a few miniature later, only gpccws is running and agents are all stopped on all nodes.  $ ps -ef | grep greenplum-cc  ~~ snip  45173 ? Tl 0:04 /usr/local/greenplum-cc-6.8.4/bin/gpccws -W masterport5432e  ~~ snip  $ su - gpadmin  $ gpcc status  2023-04-02 15:16:50 GPCC webserver: running  2023-04-02 15:16:50 GPCC agents: 0/5 agents running  2023-04-02 15:16:50 Agent is stopped on rk8-master  2023-04-02 15:16:50 Agent is stopped on rk8-slave  2023-04-02 15:16:50 Agent is stopped on rk8-node02  2023-04-02 15:16:50 Agent is stopped on rk8-node03  2023-04-02 15:16:50 Agent is stopped on rk8-node01  # When gpccws is running normally  $ curl -I --insecure --connect-timeout 5 https://localhost:28080 > /dev/null 2>&1  $ echo $?  0  $ kill -STOP <pid of gpccws>  $ curl -I --insecure --connect-timeout 5 https://localhost:28080 > /dev/null 2>&1  $ echo $?  28  $ killall postgres |
| --- |

**[48] Rollback after clearing on-fail tag on master.**

| $ pcs node standby rk8-slave  $ pcs cluster stop rk8-master  $ su - gpadmin -c "mv /data/master/gpseg-1/recovery.done /data/master/gpseg-1/recovery.conf"  $ su - gpadmin -c "export PGPORT=5432 && gpactivatestandby -d /data/master/gpseg-1/ -f"  $ su - gpadmin -c "gpinitstandby -ra"  $ su - gpadmin -c "gpinitstandby -a -s rk8-slave"  $ pcs cluster start rk8-master |
| --- |

**[49] Cleaned up gpcc on rk8-master**

| $ crm\_resource -C -r gpcc -N rk8-master -n monitor |
| --- |

**[50] Destory pacemaker cluster.**

| $ pcs resource delete gpcc  $ pcs resource delete gpdb-vip  $ pcs resource delete gpdb  $ pcs cluster disable --all  $ pcs cluster stop --all  $ pcs cluster destroy |
| --- |