triton consulting

Configuring Db2 Pacemaker HADR cluster with QDevice in AWS

By Damir Wilder

triton.co.uk

Contents

Introduction

This article will demonstrate how a Db2 Pacemaker/HADR cluster with a QDevice can be deployed on RHEL9 virtual servers in the AWS VPC (Virtual Private Cloud) environment.

The latest (currently available) version of Db2 will be used for this exercise: v11.5.9, which includes the required Pacemaker filesets.

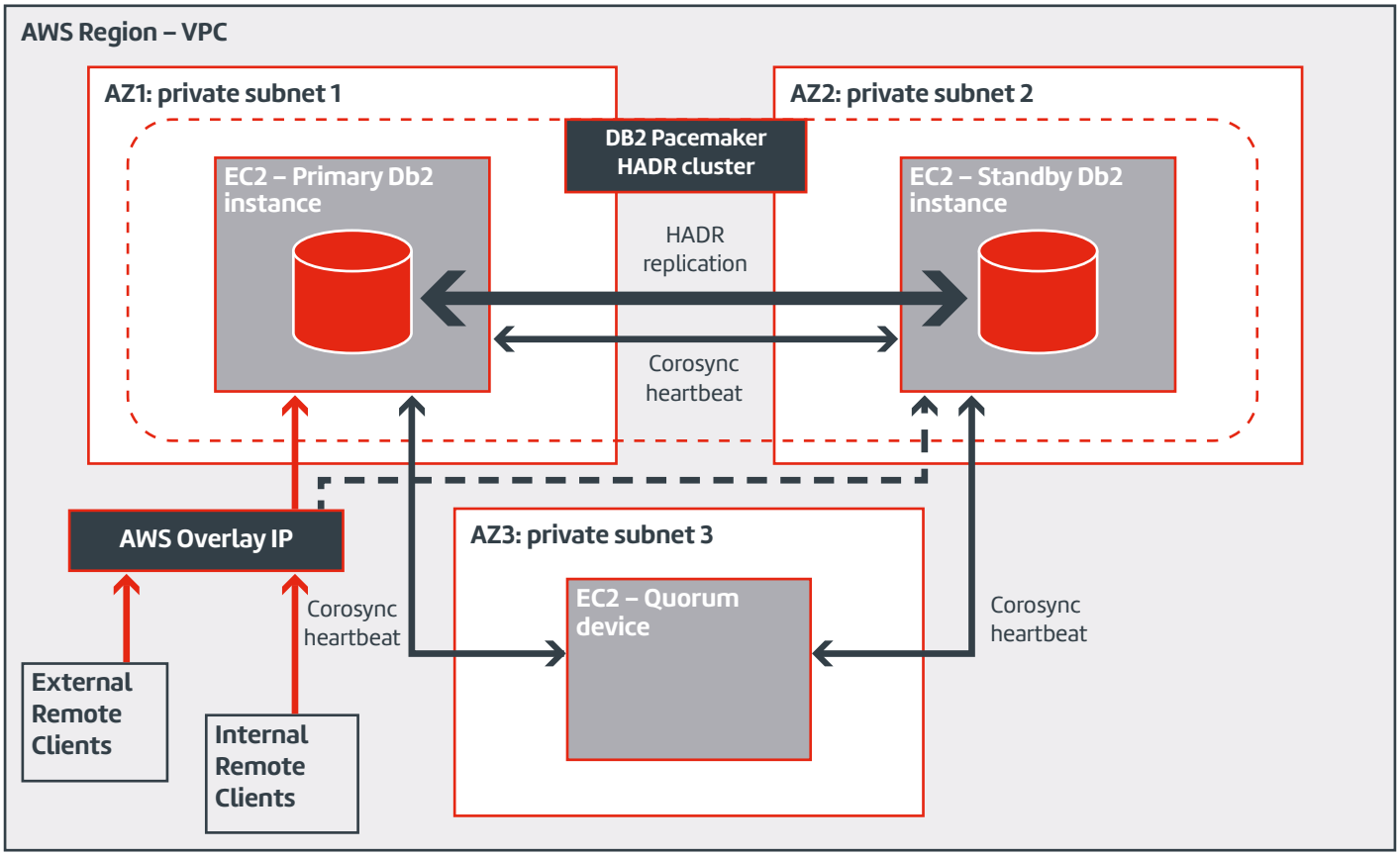
(**Note:** Pacemaker comes prepackaged with Db2 since v11.5.6, and for earlier versions it must be downloaded and installed separately).

The cluster will use a separate server as the Quorum Device, which is probably the best option for both reliability and simplicity (read more about the QDevice options in the Db2 Knowledge Centre).

Moreover, the HADR nodes will be spread across **different AZs** (Availability Zones) within the **same AWS Region**, providing availability even in a case of a whole Data Centre (AZ) failure. This will require the configuration of the **AWS Overlay IP**, which will be used as the HADR cluster's **VIP address** (i.e. the single "end point", or IP address, which all remote clients use to connect to the database, regardless of which HADR node – and in which AZ – is the current primary).

Lastly, the **ACR** (Automatic Client Reroute) will be configured and used (together with the **AWS Overlay IP**) to provide seamless and automatic client re-connection to the database in case of a failover.

When everything is done, the complete Pacemaker/HADR cluster will look like this:



Note: throughout this document, I have used the following terms interchangeably, while they all mean the same: **server == AWS instance == virtual machine == environment == platform == node**.

Also, I will assume the user AWS account has already been set up and working properly, allowing us access to the AWS VPC (Virtual Private Cloud) where all instances can and will be created (as well as all other required AWS resources), without restrictions. None of this will be discussed here, as it's out of the scope of this article.

The Prerequisites

One way to check if there are any missing prerequisites before starting the Db2 (and Pacemaker) installation (and then investigating why it failed) is to first run the **db2prereqcheck** utility, which is available as a part of the Db2 installation package.

But, in my experience, this tool tends to miss some prerequisites (and reports some which don't seem necessary), so I usually end up investigating why my Db2/Pacemaker installation failed anyway.

Therefore, I will simply list all the filesets that I had to install to get Db2 and Pacemaker installed and running OK.

Following is a list of commands I used to install the missing filesets on my Linux (RHEL9) servers:

```
yum install patch
yum install make
yum install kernel-devel
yum install gcc-c++
yum install cpp
yum install gcc
yum install ksh
yum install libaio
yum install libstdc++
yum install python3-dnf-plugin-versionlock
yum install libtool-ltdl-devel
```

Additionally, I always try to disable the SE (Security Enhanced) Linux, unless specifically told otherwise, to avoid the extra administration overhead (outside of Db2!) and the additional prerequisites and is generally a pain in the back which I like to avoid before it hits me later when I least expect it:

```
yum install grubby
grubby --update-kernel ALL --args selinux=0
```

[reboot the server now to activate the change]

The above list can be used as a reference and a starting point to sort out your missing prerequisites.

As I already mentioned in the introduction, all this was done on RHEL9 servers, so if you are running your installation on another platform, the chances are some of these filesets will already be present (and some other might be missing). Anyway, trying to install an already installed fileset will just return the information "Fileset already installed. Nothing to do." and no harm will be done.

Note: The **db2prereqcheck** utility might show two warning messages about the missing 32-bit libraries, but in my experience, these can be safely ignored (unless you need them for some exotic purpose):

```
DBT3514W  The db2prereqcheck utility failed to find the following 32-bit
library file: "/lib/libpam.so*".
DBT3514W  The db2prereqcheck utility failed to find the following 32-bit
library file: "libstdc++.so.6".
```

And another **Note**, this one on the fileset **libtool-ltdl-devel**:

When I first started setting up the Pacemaker/HADR cluster in AWS, I used RHEL8 as the underlying operating system. And everything was fine, except the fact that this version of the OS was running out of support. Then I decided to move on to the supported version, RHEL9, and suddenly this one extra fileset (**libtool-ltdl-devel**) was also reported as missing. And even worse, it was nowhere to be found – namely, it wasn't present in any of the configured default Linux Repositories. But then, after a bit of deep diving, we found the fileset in the RHEL developer's repository (codeready-builder-for-rhel-9-x86_64-rpms), which was disabled by default on the Linux servers I was working with, which is why I couldn't find the fileset in the first place. All this is well worth remembering, especially in my inexperienced-Linux-admin mind, in case it happens again.

Install Db2

When all prerequisites are (hopefully) in place, the next step is to run the system installation of Db2.

This will also install the Pacemaker filesets, which come prepackaged in the Db2 installation image (as of Db2 v11.5.6, as mentioned above).

We run the Db2 installation as follows:

```
cd [path_to_wherever_the_Db2_installation_image_was_unpacked]
db2_install -b /opt/ibm/Db2_v11.5.9 -y -f NOTSAMP
```

The switch "-f NOTSAMP" instructs the installer to not install the TSAMP files, as that won't be needed any more, with Pacemaker taking over the role of the "HA Controller" from the TSAMP.

Once again, I get those (unnecessary?) warnings about the missing 32-bit libraries, which I ignore and continue with the installation:

```
DBT3514W  The db2prereqcheck utility failed to find the following 32-bit
library file: "/lib/libpam.so*".
DBT3514W  The db2prereqcheck utility failed to find the following 32-bit
library file: "libstdc++.so.6".
```

If all prerequisites are present (and all else goes well), the installer reports the installation was successful:

```
...
The execution completed successfully.
For more information see the Db2 installation log at "/tmp/db2_install.
log.40826".
```

Missing Prerequisites

In a case where there are some prerequisites missing, the installer message will look like this:

```
...
The execution completed with warnings.
For more information see the Db2 installation log at "/tmp/db2_install.
log.80279".
```

The Db2 installation log file shows additional information on the missing stuff, for example:

```
cat /tmp/db2_install.log.80279
 ...
 Installing: PCMK
 WARNING: DBI20105E  An error occurred while installing the following file set:
    "PCMK". Because these files were not successfully installed, functionality
     that depends on these files might not work as expected.
```

To further check what is wrong with the "PCMK" (yes, this is the Pacemaker!) installation, we must look at the Pacemaker installation log file:

```
cat /tmp/db2prereqPCMK.log.44256
 ...
 The db2prereqPCMK utility found that python3-dnf-plugin-versionlock package
 is not installed on the system.
 ...
```

In this example, we just need to install the missing fileset:

```
yum install python3-dnf-plugin-versionlock
```

And then repeat the whole Db2 installation (using the same command as before), which will only install the failed "PCMK" files:

```
db2_install -b /opt/ibm/Db2_v11.5.9 -y -f NOTSAMP
...
The execution completed successfully.
For more information see the Db2 installation log at "/tmp/db2_install.
log.2315".
```

Define Db2 groups and users

Once the Db2 installation is complete, including the Pacemaker files, the next step is to create the OS groups and users which will be used by Db2:

The Db2 instance owner:

```
groupadd -g 1001 db2iadm1
useradd -u 1001 -g db2iadm1 -m -d /home/db2inst1 db2inst1
```

The Db2 fenced user:

```
groupadd -g 1002 db2fadm1
useradd -u 1002 -g db2fadm1 -m -d /home/db2fenc1 db2fenc1
```

Create Db2 Instance

When the system installation of Db2 is completed successfully, and the users/groups created, we can create the Db2 instance:

```
cd /opt/ibm/Db2_v11.5.9/instance
./db2icrt -a server -p 50000 -u db2fenc1 db2inst1
  ...
  The execution completed successfully.
  For more information see the Db2 installation log at "/tmp/db2icrt.
  log.1805".DBI1070I  Program db2icrt completed successfully.
```

To verify the Db2 instance has been created OK, we can run the **db2level** command:

```
su - db2inst1
db2level
 Db21085I  This instance or install (instance name, where applicable:
 "db2inst1") uses "64" bits and Db2 code release "SQL11059" with level
 identifier "060A010F".
 Informational tokens are "Db2 v11.5.9.0", "s2310270807",
 "DYN2310270807AMD64",and Fix Pack "0".
 Product is installed at "/opt/ibm/Db2_v11.5.9".
```

Install Pacemaker

The next step of the "system" installation is to install the Pacemaker – this is done from the filesets installed (or rather, unpacked?) during the Db2 installation.

Quick scan of the system shows where the Pacemaker installation files are located (and this will be under the directory where the Db2 installation image was unpacked to – in this case **/opt/ibm/server_dec)**

```
find / -name "db2installPCMK"
 /opt/ibm/server_dec/db2/linuxamd64/pcmk/db2installPCMK
 /opt/ibm/server_dec/db2/linuxamd64/bin/pcmk/db2installPCMK
```

Next, we run the Pacemaker installation from any of the above directories:

```
/opt/ibm/server_dec/db2/linuxamd64/pcmk/db2installPCMK -i
  Installing "Pacemaker"
  Success
  DBI1070I  Program db2installPCMK completed successfully.
```

To confirm the installation, we can check if the Pacemaker Daemon executable is present on the server:

```
find / -name "pacemakerd"
  /usr/sbin/pacemakerd
  /usr/src/debug/pacemaker-2.1.6-4.db2pcmk.el8.x86_64/daemons/pacemaker
```

Install AWS CLI

We must not forget to install the AWS CLI on the Db2 servers as well, which will be required later on for the configuration of the AWS Overlay IP resource.

The best (and most up-to-date) way to do this is to follow the detailed AWS documentation.

When we're done, the "aws" command should work from the command line:

```
aws --version
 aws-cli/2.7.4 Python/3.9.11 Linux/4.18.0-513.9.1.el8_9.x86_64 exe/x86_64.
 rhel.8
```

Clone the Servers

Db2 HADR Nodes

- after the (system) installation of the first node is complete, as described in the above steps, either repeat the same process step by step on the second node (and all other nodes, if aiming for a HADR cluster with more than one Standby), which can be tedious, or do it the "cloudy" way: clone the server by shutting it down; taking a snapshot of it (see here: AMI – Amazon Machine Image); and then creating a new AWS instance (node) from that snapshot. That's quick, nice, and easy, just as we like it!

QDevice

- this server does not need all Db2 installation files, just the **corosync-qnetd** RPM (which can be found on an existing Db2 node and copied via SSH – see the details here), so can be created clean from the AWS console (not using the Db2 node snapshot as described above).

When all required servers have been cloned or step-by-step configured, we can proceed with setting up the rest of the system resources (which will differ from server to server), the Db2 resources, the HADR cluster and the Pacemaker cluster.

Set up Passwordless SSH

The passwordless SSH must be enabled between all cluster nodes for the **root** user and the Db2 instance owner user ID (**db2inst1**).

Root user must also be able to use the passwordless SSH between all cluster nodes and the QDevice server.

This is done by exchanging the public SSH keys between **root** (and **db2inst1**) user accounts. Once again, in the interests of saving the space, I will not go into the details here but point you to decent documentation on how this can be done.

From my experience, here are a few notes worth remembering when setting up the passwordless SSH:

- leave the SSH passphrase empty.
- probably a good idea to remove the banners (/etc/motd, /etc/ssh/my_banner, ...) straight away, so that they don't interfere with the Pacemaker configuration later.
- copy the root's public SSH key to the local **authorized_keys** file as well, Pacemaker seems to need this when configuring the cluster.

Set up Hosts Files

The host's file is a Linux system file located in the /etc directory of each host.

This file must be edited (on all nodes) to include the relevant information for all cluster nodes, including the QDevice: IP address, long server name (Fully Qualified Domain Name), short server name (alias).

The instance user and root ID must be able to use SSH between the two hosts using both Fully Qualified Domain Name and hostname aliases.

Note: Editing the local hosts file is probably not necessary if the above can be handled by the DNS, but I've included it here for completeness (not all of us know how to setup DNS, or have access to it!), and because it is mentioned in the documentation.

Create the Db2 Database

At this point we are ready to create the Db2 database, which will then be set up for the high availability and disaster recovery.

In this example we will create the SAMPLE database by running the following commands on the "primary-to-be" node, logged in as the Db2 instance owner (**db2inst1**):

```
cd ~/sqllib/bin
./db2sampl
```

In real life, we will likely want to use an existing database (or restore a database from a backup image) and there is nothing wrong with that, the rest of the process remains the same.

Configure the Db2 Database for HADR

Once the database has been created (from the scratch, from the backup, or else...), we must configure it for the log archiving, if this hasn't been done already:

```
mkdir ~/logarchive
db2 "update db cfg for SAMPLE using LOGARCHMETH1 'disk:/home/db2inst1/
logarchive'"
db2 backup db SAMPLE
```

Then, we take a full DB backup and transfer it to the "standby-to-be" node:

```
db2 backup db SAMPLE online include logs
scp SAMPLE.0...20231213143617.001 db2inst1@hadr_stby:/home/db2inst1
```

On the "standby-to-be" node, we proceed to restore the database:

```
mkdir ~/logarchive
db2 restore db SAMPLE from . taken at 20231213143617
    Db20000I  The RESTORE DATABASE command completed successfully.
```

The last step to do is configure the HADR parameters:

– on the Primary:

```
db2 "update db cfg for SAMPLE using HADR_LOCAL_HOST hadr_prim"
db2 "update db cfg for SAMPLE using HADR_LOCAL_SVC 50100"
db2 "update db cfg for SAMPLE using HADR_REMOTE_HOST hadr_stby"
db2 "update db cfg for SAMPLE using HADR_REMOTE_SVC 50100"
db2 "update db cfg for SAMPLE using HADR_REMOTE_INST db2inst1"
db2 "update db cfg for SAMPLE using HADR_TARGET_LIST 'hadr_stby:50100'"
db2 "update db cfg for SAMPLE using HADR_PEER_WINDOW 120"
```

– on the Standby:

```
db2 "update db cfg for SAMPLE using HADR_LOCAL_HOST hadr_stby"
db2 "update db cfg for SAMPLE using HADR_LOCAL_SVC 50100"
db2 "update db cfg for SAMPLE using HADR_REMOTE_HOST hadr_prim"
db2 "update db cfg for SAMPLE using HADR_REMOTE_SVC 50100"
db2 "update db cfg for SAMPLE using HADR_REMOTE_INST db2inst1"
db2 "update db cfg for SAMPLE using HADR_TARGET_LIST 'hadr_prim:50100'"
```

Now we are ready to start the HADR on the Primary and the Standby:

First the Standby:

```
db2 start HADR on db SAMPLE as standby
 Db20000I  The START HADR ON DATABASE command completed successfully.
```

Then the Primary:

```
db2 start HADR on db SAMPLE as primary
 Db20000I  The START HADR ON DATABASE command completed successfully.
```

When both commands complete (without errors), we can check the HADR status by running the following command (on either server):

```
db2pd -d SAMPLE -hadr
```

So, now our database is running within a HADR cluster, which means if the active server (Primary) fails, the Standby server will still contain a complete and up-to-date copy of the database. But the failover must be initiated by hand, HADR does not do this for us. Therefore, we have up until now covered only the disaster recovery (DR) aspect of the HADR.

To get the high availability (HA) feature as well (that is, fully automated failovers), we must configure the Pacemaker cluster. We will do this next.

Configure the Pacemaker Cluster

The required Pacemaker resources are created by running the command **db2cm** with appropriate options, under the **root** user account. This command can be executed on any node (Primary, Standby) in the cluster (excluding the QDevice node, unless it was made a part of the Pacemaker cluster):

– the cluster and the public network resources:

```
cd /home/db2inst1/sqllib/bin/
./db2cm -create -cluster -domain PCM_Domain -host hadr_prim -publicEthernet
eth0 -host hadr_stby -publicEthernet eth0
    Created db2_hadr_prim_eth0 resource.
    Created db2_hadr_stby_eth0 resource.
    Cluster created successfully.
```

– the instance resource, for Primary:

```
./db2cm -create -instance db2inst1 -host hadr_prim
    Created db2_hadr_prim_db2inst1_0 resource.
    Instance resource for db2inst1 on hadr_prim created successfully.
```

– the instance resource, for Standby:

```
./db2cm -create -instance db2inst1 -host hadr_stby
    Created db2_hadr_stby_db2inst1_0 resource.
    Instance resource for db2inst1 on hadr_stby created successfully.
```

– the database resource:

```
./db2cm -create -db SAMPLE -instance db2inst1
    Database resource for SAMPLE created successfully.
```

With all this, the Db2 database (i.e. the whole HADR cluster) has been put under the Pacemaker control.
To verify this, run the following command (on all HADR nodes):

```
db2 get dbm cfg | grep -i cluster
```

The output should look like this (if it doesn't, something went wrong somewhere...):

```
Cluster manager    = Pacemaker
```

The Pacemaker cluster status can be checked with:

```
crm status
```

And it should show all above defined resources to be up and running (green colour).

Now, if anything happens with the Primary node and it fails, the failover to the Standby node will be automatic and it will assume the role of the primary node. However, the clients will still need to be reconfigured (manually?!) to access the Standby node via a different IP address.

To make all this even better, we will create the VIP resource next and then ACR as well to fully automate the failovers.

Create the VIP resource using the AWS Overlay IP address

In AWS, if we want to spread our HADR nodes across different AZs (as we did in this example), we cannot simply configure the "standard" VIP address in the Pacemaker as we are used to, because it won't be able to handle the routing across the Availability Zones.

Instead, we must use the AWS Overlay IP address as the HADR VIP resource, which is capable of just that – routing the IP traffic across the different AZs.

There are a few "prerequisite" steps that must be taken to get this done:

1. Decide on an (unused) IP address and a network interface that will be used for this purpose:

   <OVERLAY_IP>
   <ETHx>

   Note: all hosts must use a network interface with the same name (for example: **eth1**)
2. Disable "source/destination check" for the EC2 instances hosting the IBM Db2 primary and standby database:

   –go to the AWS Console, select the EC2 Instance in the EC2 Management Console, then select "Actions" and "Networking" and set "Change source / destination checking" to "stop"
3. Create a policy and attach it to your IAM role by using the AWS IAM Management Console: **<ROUTE_TABLE_ID>**

Now, on the command line, we create a profile by using the **aws configure** command:

– for this, we need to interactively type in the AWS Access Key ID and the Secret Access Key, as well as the AWS region where our instances are located, so have this information ready! (find more details about the AWS keys here):

```
aws configure --profile db2-user
  AWS Access Key ID [None]: <AWS_access_key_ID>
  AWS Secret Access Key [None]: <AWS_secret_access_key>
  Default region name [None]: eu-west-2
  Default output format [None]:
```

If we got everything right, the profile is created, and we can check it with:

```
aws s3 ls --profile db2-user
  2024-01-19 16:28:30 stackset-triton-aws-config-a94a7a1e-configbuck-
et-1jisngsyqlsxa
```

Next, we run a set of commands to update the routing tables (one for each AZ!) with the Overlay IP pointing to the node with the Db2 primary instance (the **route-table-ids** can be found in the AWS Console, as well as the AWS ID of the Primary instance):

```
aws ec2 create-route --route-table-id <rtb-AZ-01> --destination-cidr-block
<OVERLAY_IP> --instance-id <Primary_AWS_Inst_ID> --profile db2-user
  {
      "Return": true
  }
aws ec2 create-route --route-table-id <rtb-AZ-02> --destination-cidr-block
<OVERLAY_IP> --instance-id <Primary_AWS_Inst_ID> --profile db2-user
  {
      "Return": true
  }
aws ec2 create-route --route-table-id <rtb-AZ-03> --destination-cidr-block
<OVERLAY_IP> --instance-id <Primary_AWS_Inst_ID> --profile db2-user
  {
      "Return": true
  }
```

Finally, we can now create the Overlay IP resource itself (notice the same set of routing table IDs here, as in the above "create-route" commands):

```
db2cm -create -aws -primaryVIP <OVERLAY_IP> -rtb <rtb-AZ-01>,<rtb-AZ-02>,
<rtb-AZ-03> -profile db2-user -db SAMPLE -instance db2inst1
  Created overlay IP resource db2_db2inst1_db2inst1_SAMPLE-primary-OIP
  for SAMPLE primary successfully.
```

At this point, if everything went smoothly, we should have our AWS Overlay VIP address available, and from now on it will serve as a single "end point" which all remote clients (both outside and within AWS) will use to connect to the SAMPLE database.

We can check the Pacemaker cluster status, together with the just now created AWS Overlay VIP (highlighted in red colour, below) and all other defined resources, with the following command:

```
crm status
  Cluster Summary:
    * Stack: corosync (Pacemaker is running)
    * Current DC: ip-hadr-stby (version 2.1.6-4.db2pcmk.el9-6fdc9deea29) -
partition with quorum
    * Last updated: Fri Jan 19 17:09:07 2024 on ip-hadr-prim
    * Last change:  Fri Jan 19 17:08:37 2024 by root via cibadmin on ip-hadr-
prim
    * 2 nodes configured
    * 9 resource instances configured

  Node List:
    * Online: [ ip-hadr-prim ip-hadr-stby ]

  Full List of Resources:
   * db2_ip-hadr-prim_eth0     (ocf:heartbeat:db2ethmon):       Started ip-
hadr-prim
   * db2_ip-hadr-stby_eth0     (ocf:heartbeat:db2ethmon):       Started ip-
hadr-stby
   * db2_ip-hadr-prim_db2inst1_0  (ocf:heartbeat:db2inst):      Started ip-
hadr-prim
   * db2_ip-hadr-stby_db2inst1_0  (ocf:heartbeat:db2inst):      Started ip-
hadr-stby
   * Clone Set: db2_db2inst1_db2inst1_SAMPLE-clone [db2_db2inst1_db2inst1_SAM-
PLE] (promotable):
     * Promoted: [ ip-hadr-prim ]
     * Unpromoted: [ ip-hadr-stby ]
   * db2_db2inst1_db2inst1_SAMPLE-primary-OIP (ocf:heartbeat:aws-vpc-move-ip):
Started ip-hadr-prim
```

Note: For more information on the AWS Overlay IP prerequisites and setup, check out this document!

Configure the ACR (Automatic Client Reroute)

With the "single end point" in place, we can now configure the Automatic Client Reroute, which will enable the Db2 clients to automatically reconnect to the database in a case of a failover.

This is done by setting up both HADR servers to use the defined <OVERLAY_IP> as the alternate server hostname:

On Primary:

```
db2 update alternate server for database SAMPLE
    using hostname <OVERLAY_IP> port 50000
```

On Standby:

```
db2 update alternate server for database SAMPLE
    using hostname <OVERLAY_IP> port 50000
```

To check the configuration has been updated successfully, run the following command (output slightly edited to only show the relevant info – in red colour):

```
db2 list db directory
  ...
Database 1 entry:
   Database alias                       = SAMPLE
   Database name                        = SAMPLE
   Local database directory             = /home/db2inst1
   Database release level               = 15.00
   Alternate server hostname            = <OVERLAY_IP>
   Alternate server port number         = 50000
```

The change is automatic and there is no need to restart the database or the instance. Only the already connected clients will need to be reconnected to become aware of the change.

Now, it would be a good idea to test the client connections' resiliency during a failover, to confirm they can survive the event without any external help, and don't need any explicit reconnections to the database.

I will not repeat this testing here, as I have already done it, but will instead point you to my blog post where I've described the testing in detail.

Install and configure a QDevice Quorum

Up until now, the Pacemaker has been using the default "Two-node" quorum, which is prone to split-brain scenarios and is not meant to be used in production environments.

So, let's take care of this now and set up a QDevice Quorum on a physically separate server!

On the Primary and Standby hosts, ensure that the corosync-qdevice package is installed:

```
rpm -qa | grep corosync-qdevice
    corosync-qdevice-debugsource-3.0.3-1.db2pcmk.el9.x86_64
    corosync-qdevice-debuginfo-3.0.3-1.db2pcmk.el9.x86_64
    corosync-qdevice-3.0.3-1.db2pcmk.el9.x86_64
    corosync-qdevice-devel-3.0.3-1.db2pcmk.el9.x86_64
```

If it is not already installed, we can install it now, taking care to specify the correct path to the installation packages:

```
dnf install <Db2_image>/db2/<platform>/pcmk/Linux/<OS_distribution>/
<architecture>/corosync-qdevice*
```

In my example, the command looks like this:

```
dnf install /opt/ibm/server_dec/db2/linuxamd64/install/pcmk/Linux/rhel/rhel9/
x86_64/corosync-qdevice*
```

But the files are already installed on my Db2 servers, so nothing to be done here.

On the third host (Q-DEVICE), we must install the Corosync QNet software. Again, we need to specify the correct path to the installation packages, such as:

```
dnf install /opt/ibm/server_dec/db2/linuxamd64/install/pcmk/Linux/rhel/rhel9/
x86_64/corosync-qnetd*
```

The installation can be checked with:

```
rpm -qa | grep corosync-qnetd
    corosync-qnetd-debuginfo-3.0.3-1.db2pcmk.el9.x86_64
    corosync-qnetd-3.0.3-1.db2pcmk.el9.x86_64
```

Finally, as the root user, we run the db2cm command to setup the QDevice from one of the cluster nodes (Primary, Standby):

```
home/db2inst1/sqllib/bin/db2cm -create -qdevice 172.18.19.90
    Successfully configured qdevice on nodes ip-hadr-prim and ip-hadr-stby
    Attempting to start qdevice on <Q-DEVICE-IP>
    Quorum device <Q-DEVICE-IP> added successfully.
```

At this point, the setup should be complete, and the Pacemaker cluster should have switched from the default “two-node quorum” to the more reliable QDevice quorum!

To verify the QDevice setup, we can run the following commands:

On the Primary and Standby:

```
corosync-qdevice-tool -s

    Qdevice information
    -------------------
    Model:                  Net
    Node ID:                2
    Configured node list:
        0   Node ID = 1
        1   Node ID = 2
    Membership node list:   1, 2

    Qdevice-net information
    ----------------------
    Cluster name:           PCM_Domain
    QNetd host:             <Q-DEVICE-IP>:5403
    Algorithm:              LMS
    Tie-breaker:            Node with lowest node ID
    State:                  Connected
```

On the QDevice host:

```
corosync-qnetd-tool -l

    Cluster " PCM_Domain":
        Algorithm:          LMS
        Tie-breaker:        Node with lowest node ID
        Node ID 1:
            Client address:         ::ffff:<hadr-primary-IP>:39642
            Configured node list:   1, 2
            Membership node list:   1, 2
            Vote:                   ACK (ACK)
        Node ID 2:
            Client address:         ::ffff:<hadr-standby-IP>:60674
            Configured node list:   1, 2
            Membership node list:   1, 2
            Vote:                   ACK (ACK)
```

Corosync Heartbeat Hardening

One last thing that we may want to do to make our Pacemaker cluster even more robust is to enhance its network resiliency by setting up additional heartbeat rings.

The idea here is to set up every heartbeat ring to operate on a separate network interface (and separate network infrastructure as well), to avoid any single points of failure in the network.

I haven't tested this concept, because I didn't have the required network infrastructure in place but would certainly think very hard about provisioning it for any production and mission-critical environments.

If you're interested in this concept and want to try it yourself, find the relevant info here or contact us !