Technical Guide

# High availability features of DB Service

September 2021

SAMSUNG SDS

# Contents

# 1. Overview

The purpose of this document is to describe the AutoRestart function among the high availability features provided by DB Service. The HA features increase the availability of a DB consisting of a single virtual server in a cloud environment.

The AutoRestart function raises the availability of a single DB server by utilizing the high availability (HA) function of the virtual server through hypervisor clustering and the resource management function of a redundancy solution Pacemaker.

# 2. High availability from hypervisor clustering

A virtual server runs on a hypervisor through which hardware resources are allocated to the virtual server. If a hypervisor or a physical host server fails, the virtual server running on the hypervisor will also fail.

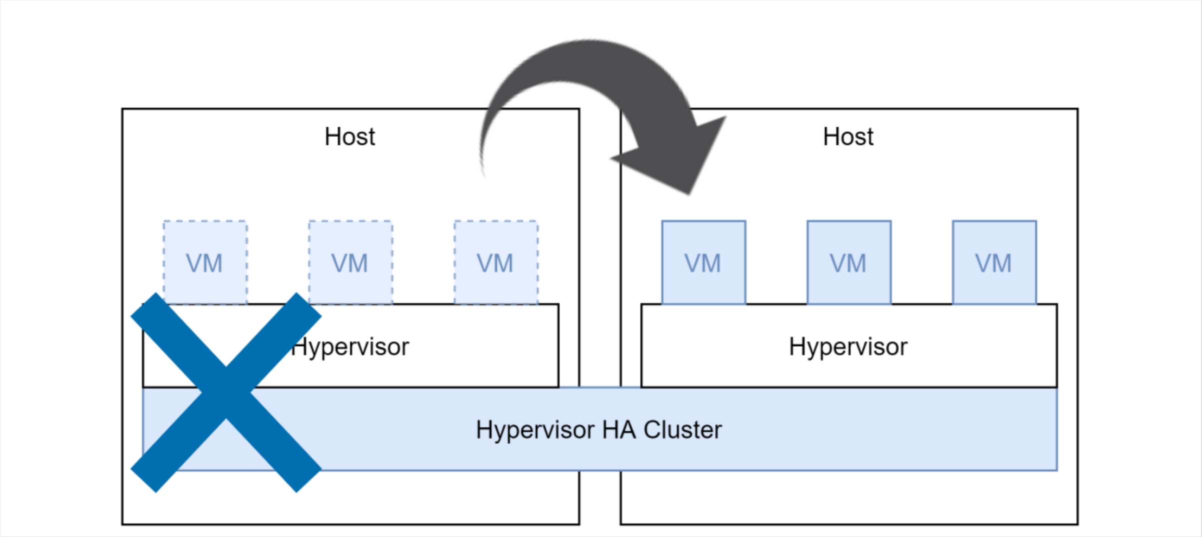


**Figure 1. Virtual Server HA with hypervisor clustering**

To prevent this, hypervisors are installed on different physical host servers and hypervisor clusters for HA are configured. This allows the virtual server running on a certain hypervisor to continue service provision in the event of a problem with the hypervisor or physical host server by moving it to another one.

# 3. Resource management of redundancy solution

The abovementioned virtual server HA from hypervisor clustering is a method to increase the availability of the virtual server itself against hardware-level failure but it is unable to process OS, DBMS, or other applications within the virtual server.

For instance , the OS restarts when moving a virtual server but the service may not be provided unless an automatic start of DBMS and other applications is set separately when the OS is booted.

To compensate for this, Pacemaker can be used to configure a single node cluster and register the DB as a resource in the resource management function to manage monitoring, start, stop, restart and more.

The redundancy solution also allows DB resources to start automatically when the OS is booted while enabling process monitoring and SQL query-level monitoring for DB resources, detecting DB failures and conducting restarts.

## 4. Components of Pacemaker

The following are the main components of Pacemaker that serve resource management functions.

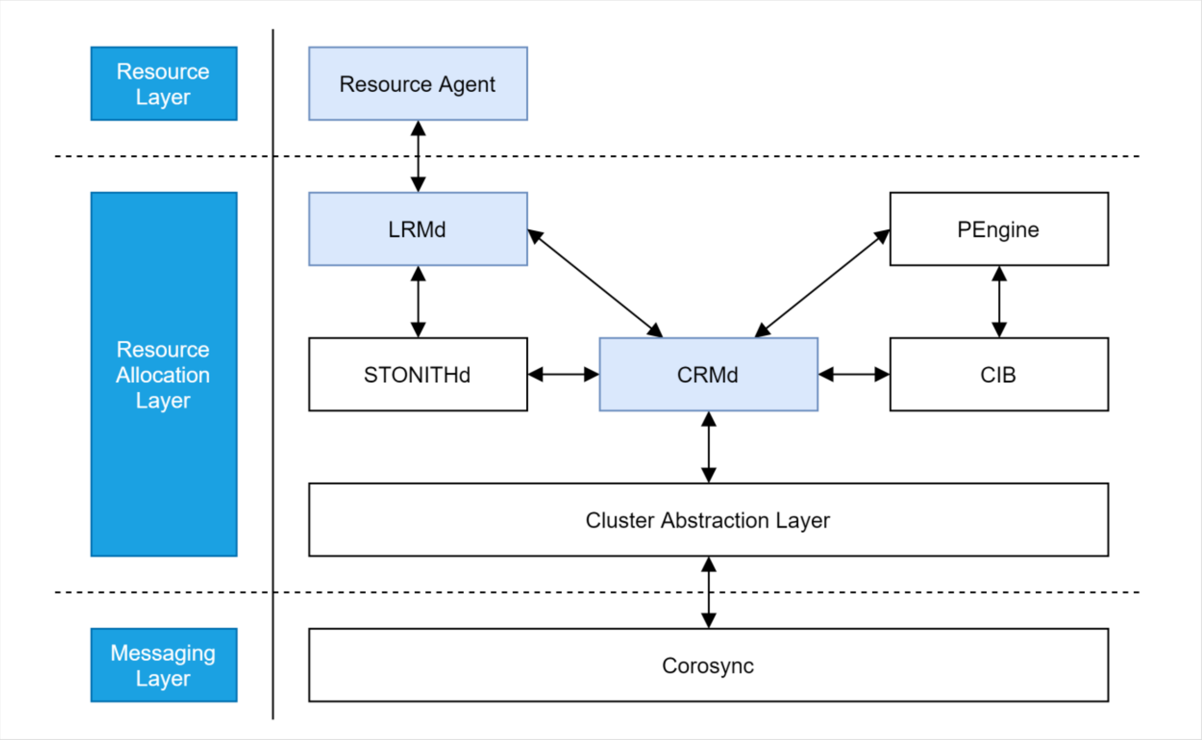


**Figure 2. Components of Pacemaker**

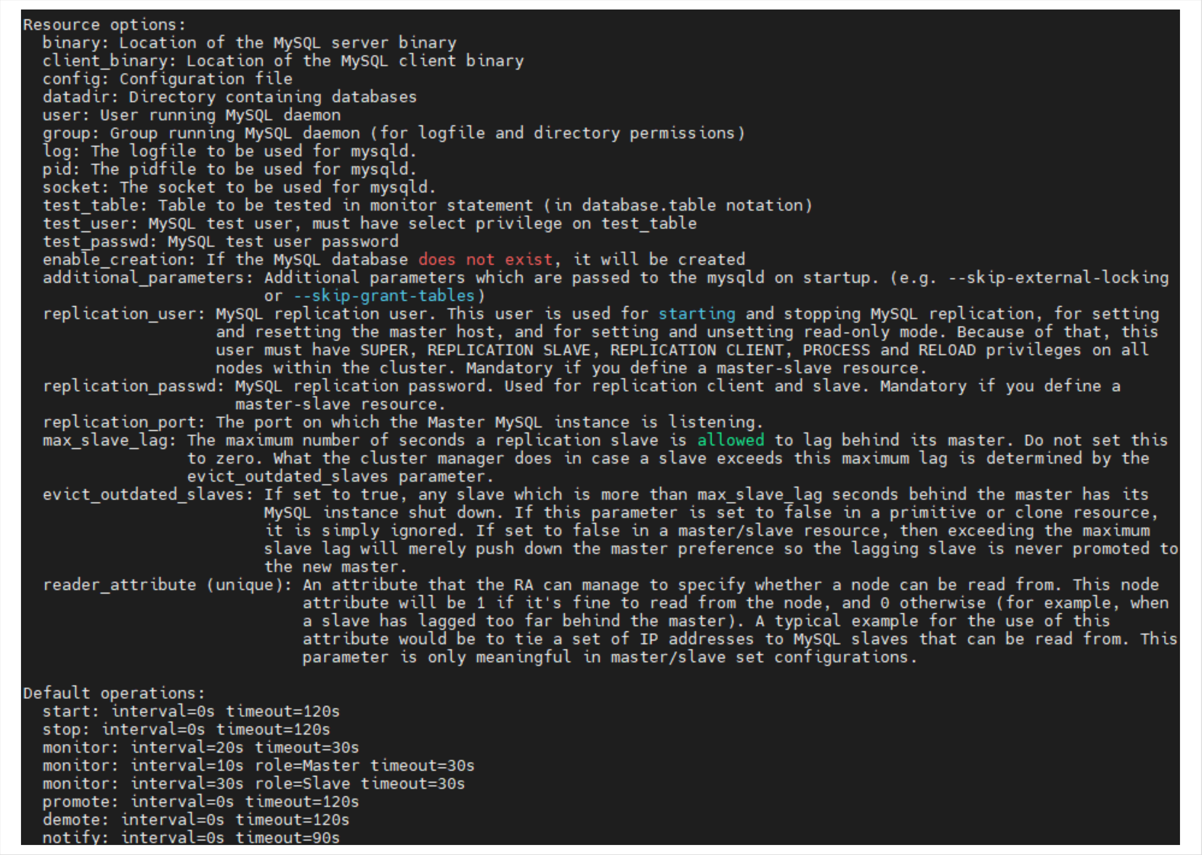
1. CRMd (Cluster Resource Manager daemon): Serve the main controller process role, routing all resource operations and handling all operations within the resource allocation layer
2. LRMd (Local Resource Manager daemon): Act as an interface between CRMd and Resource Agent
3. Resource Agent: A standardized interface defined for cluster resources,

running scripts such as start/stop/monitor. There are various agents suitable for different applications.

4. CIB (Cluster Information Base): Manage configuration information and set as XML files
5. PEngine (Policy Engine): Manage policies and check dependencies when switching resources
6. STONITHd: Fencing Agent
7. Corosync: A messaging system used in cluster environment, which is basic infrastructure required for Pacemaker operation

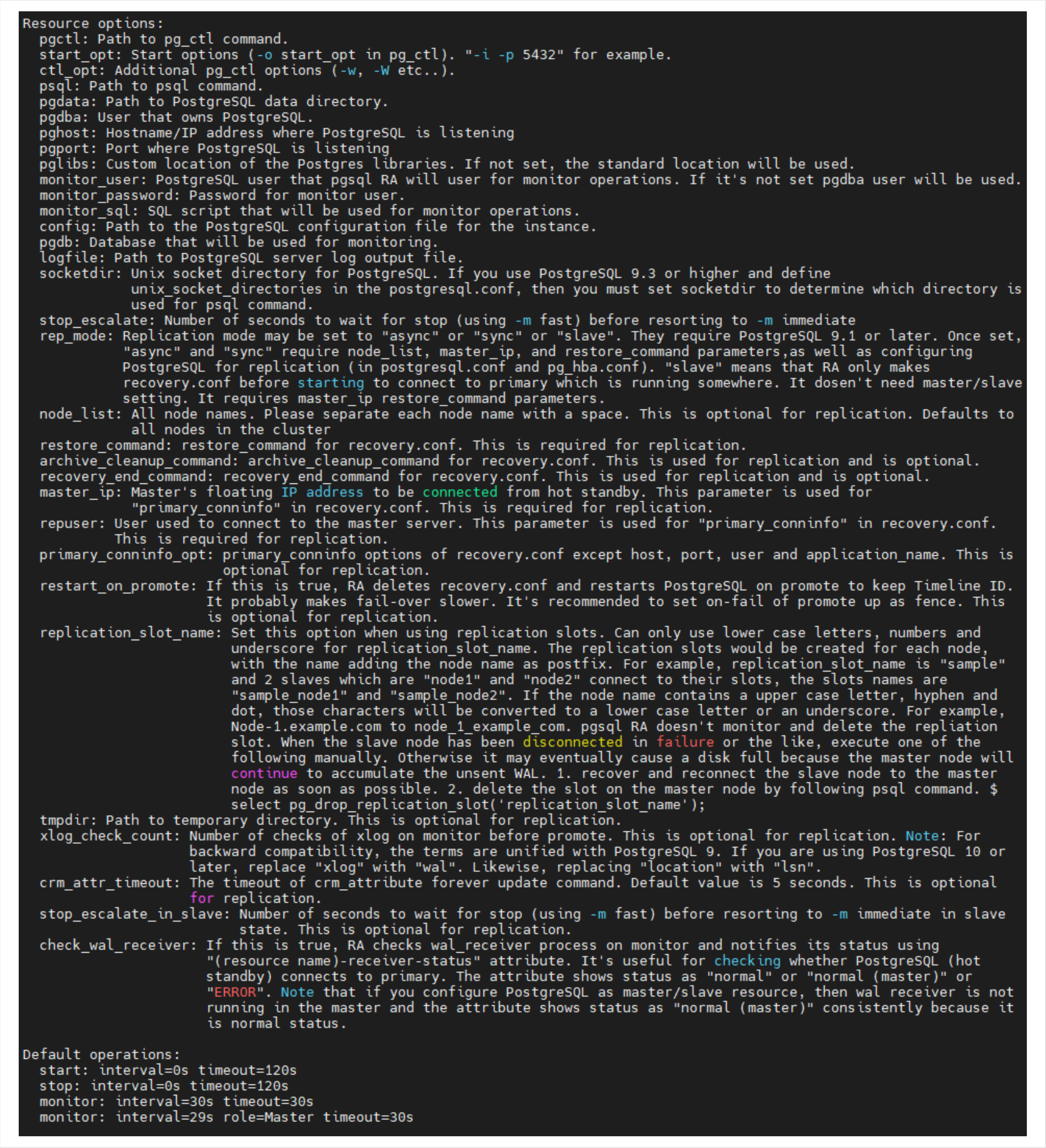
## 5. Configuration options for Pacemaker DB resources

The options to configure Pacemaker's resources vary by resource agent. The resource configuration options for leading open source DBs MySQL and PostgreSQL are as follows:



```
Resource options:
  binary: Location of the MySQL server binary
  client_binary: Location of the MySQL client binary
  config: Configuration file
  datadir: Directory containing databases
  user: User running MySQL daemon
  group: Group running MySQL daemon (for logfile and directory permissions)
  log: The logfile to be used for mysqld.
  pid: The pidfile to be used for mysqld.
  socket: The socket to be used for mysqld.
  test_table: Table to be tested in monitor statement (in database.table notation)
  test_user: MySQL test user, must have select privilege on test_table
  test_passwd: MySQL test user password
  enable_creation: If the MySQL database does not exist, it will be created
  additional_parameters: Additional parameters which are passed to the mysqld on startup. (e.g. --skip-external-locking
                         or --skip-grant-tables)
  replication_user: MySQL replication user. This user is used for starting and stopping MySQL replication, for setting
                    and resetting the master host, and for setting and unsetting read-only mode. Because of that, this
                    user must have SUPER, REPLICATION SLAVE, REPLICATION CLIENT, PROCESS and RELOAD privileges on all
                    nodes within the cluster. Mandatory if you define a master-slave resource.
  replication_passwd: MySQL replication password. Used for replication client and slave. Mandatory if you define a
                      master-slave resource.
  replication_port: The port on which the Master MySQL instance is listening.
  max_slave_lag: The maximum number of seconds a replication slave is allowed to lag behind its master. Do not set this
                 to zero. What the cluster manager does in case a slave exceeds this maximum lag is determined by the
                 evict_outdated_slaves parameter.
  evict_outdated_slaves: If set to true, any slave which is more than max_slave_lag seconds behind the master has its
                         MySQL instance shut down. If this parameter is set to false in a primitive or clone resource,
                         it is simply ignored. If set to false in a master/slave resource, then exceeding the maximum
                         slave lag will merely push down the master preference so the lagging slave is never promoted to
                         the new master.
  reader_attribute (unique): An attribute that the RA can manage to specify whether a node can be read from. This node
                             attribute will be 1 if it's fine to read from the node, and 0 otherwise (for example, when
                             a slave has lagged too far behind the master). A typical example for the use of this
                             attribute would be to tie a set of IP addresses to MySQL slaves that can be read from. This
                             parameter is only meaningful in master/slave set configurations.

Default operations:
  start: interval=0s timeout=120s
  stop: interval=0s timeout=120s
  monitor: interval=20s timeout=30s
  monitor: interval=10s role=Master timeout=30s
  monitor: interval=30s role=Slave timeout=30s
  promote: interval=0s timeout=120s
  demote: interval=0s timeout=120s
  notify: interval=0s timeout=90s
```

**Figure 3. Resource options of MySQL DBMS**

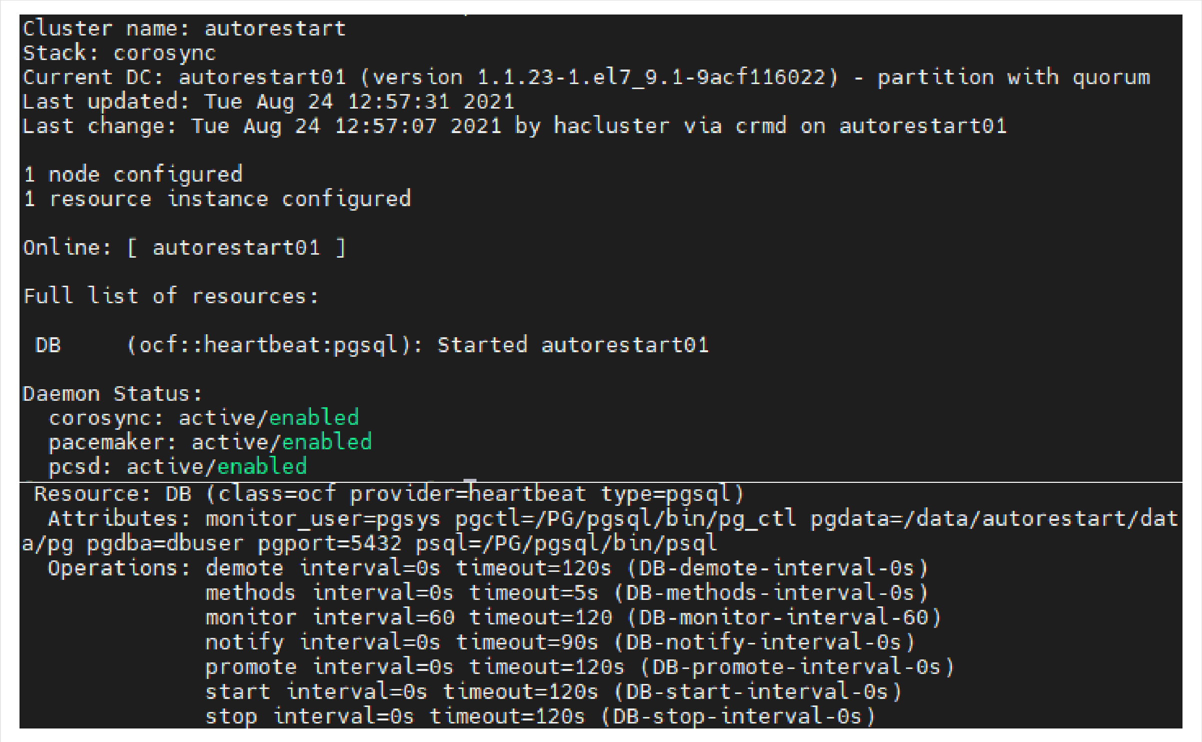


```
Resource options:
  pgctl: Path to pg_ctl command.
  start_opt: Start options (-o start_opt in pg_ctl). "-i -p 5432" for example.
  ctl_opt: Additional pg_ctl options (-w, -W etc..).
  psql: Path to psql command.
  pgdata: Path to PostgreSQL data directory.
  pgdba: User that owns PostgreSQL.
  pghost: Hostname/IP address where PostgreSQL is listening
  pgport: Port where PostgreSQL is listening
  pglibs: Custom location of the Postgres libraries. If not set, the standard location will be used.
  monitor_user: PostgreSQL user that pgsql RA will user for monitor operations. If it's not set pgdba user will be used.
  monitor_password: Password for monitor user.
  monitor_sql: SQL script that will be used for monitor operations.
  config: Path to the PostgreSQL configuration file for the instance.
  pgdb: Database that will be used for monitoring.
  logfile: Path to PostgreSQL server log output file.
  socketdir: Unix socket directory for PostgreSQL. If you use PostgreSQL 9.3 or higher and define
             unix_socket_directories in the postgresql.conf, then you must set socketdir to determine which directory is
             used for psql command.
  stop_escalate: Number of seconds to wait for stop (using -m fast) before resorting to -m immediate
  rep_mode: Replication mode may be set to "async" or "sync" or "slave". They require PostgreSQL 9.1 or later. Once set,
            "async" and "sync" require node_list, master_ip, and restore_command parameters,as well as configuring
            PostgreSQL for replication (in postgresql.conf and pg_hba.conf). "slave" means that RA only makes
            recovery.conf before starting to connect to primary which is running somewhere. It dosen't need master/slave
            setting. It requires master_ip restore_command parameters.
  node_list: All node names. Please separate each node name with a space. This is optional for replication. Defaults to
             all nodes in the cluster
  restore_command: restore_command for recovery.conf. This is required for replication.
  archive_cleanup_command: archive_cleanup_command for recovery.conf. This is used for replication and is optional.
  recovery_end_command: recovery_end_command for recovery.conf. This is used for replication and is optional.
  master_ip: Master's floating IP address to be connected from hot standby. This parameter is used for
             "primary_conninfo" in recovery.conf. This is required for replication.
  repuser: User used to connect to the master server. This parameter is used for "primary_conninfo" in recovery.conf.
           This is required for replication.
  primary_conninfo_opt: primary_conninfo options of recovery.conf except host, port, user and application_name. This is
                        optional for replication.
  restart_on_promote: If this is true, RA deletes recovery.conf and restarts PostgreSQL on promote to keep Timeline ID.
                      It probably makes fail-over slower. It's recommended to set on-fail of promote up as fence. This
                      is optional for replication.
  replication_slot_name: Set this option when using replication slots. Can only use lower case letters, numbers and
                         underscore for replication_slot_name. The replication slots would be created for each node,
                         with the name adding the node name as postfix. For example, replication_slot_name is "sample"
                         and 2 slaves which are "node1" and "node2" connect to their slots, the slots names are
                         "sample_node1" and "sample_node2". If the node name contains a upper case letter, hyphen and
                         dot, those characters will be converted to a lower case letter or an underscore. For example,
                         Node-1.example.com to node_1_example_com. pgsql RA doesn't monitor and delete the repliation
                         slot. When the slave node has been disconnected in failure or the like, execute one of the
                         following manually. Otherwise it may eventually cause a disk full because the master node will
                         continue to accumulate the unsent WAL. 1. recover and reconnect the slave node to the master
                         node as soon as possible. 2. delete the slot on the master node by following psql command. $
                         select pg_drop_replication_slot('replication_slot_name');
  tmpdir: Path to temporary directory. This is optional for replication.
  xlog_check_count: Number of checks of xlog on monitor before promote. This is optional for replication. Note: For
                    backward compatibility, the terms are unified with PostgreSQL 9. If you are using PostgreSQL 10 or
                    later, replace "xlog" with "wal". Likewise, replacing "location" with "lsn".
  crm_attr_timeout: The timeout of crm_attribute forever update command. Default value is 5 seconds. This is optional
                    for replication.
  stop_escalate_in_slave: Number of seconds to wait for stop (using -m fast) before resorting to -m immediate in slave
                          state. This is optional for replication.
  check_wal_receiver: If this is true, RA checks wal_receiver process on monitor and notifies its status using
                      "(resource name)-receiver-status" attribute. It's useful for checking whether PostgreSQL (hot
                      standby) connects to primary. The attribute shows status as "normal" or "normal (master)" or
                      "ERROR". Note that if you configure PostgreSQL as master/slave resource, then wal receiver is not
                      running in the master and the attribute shows status as "normal (master)" consistently because it
                      is normal status.

Default operations:
  start: interval=0s timeout=120s
  stop: interval=0s timeout=120s
  monitor: interval=30s timeout=30s
  monitor: interval=29s role=Master timeout=30s
```

**Figure 4. Resource options of PostgreSQL DBMS**

# 6. Test results of Pacemaker DB resource

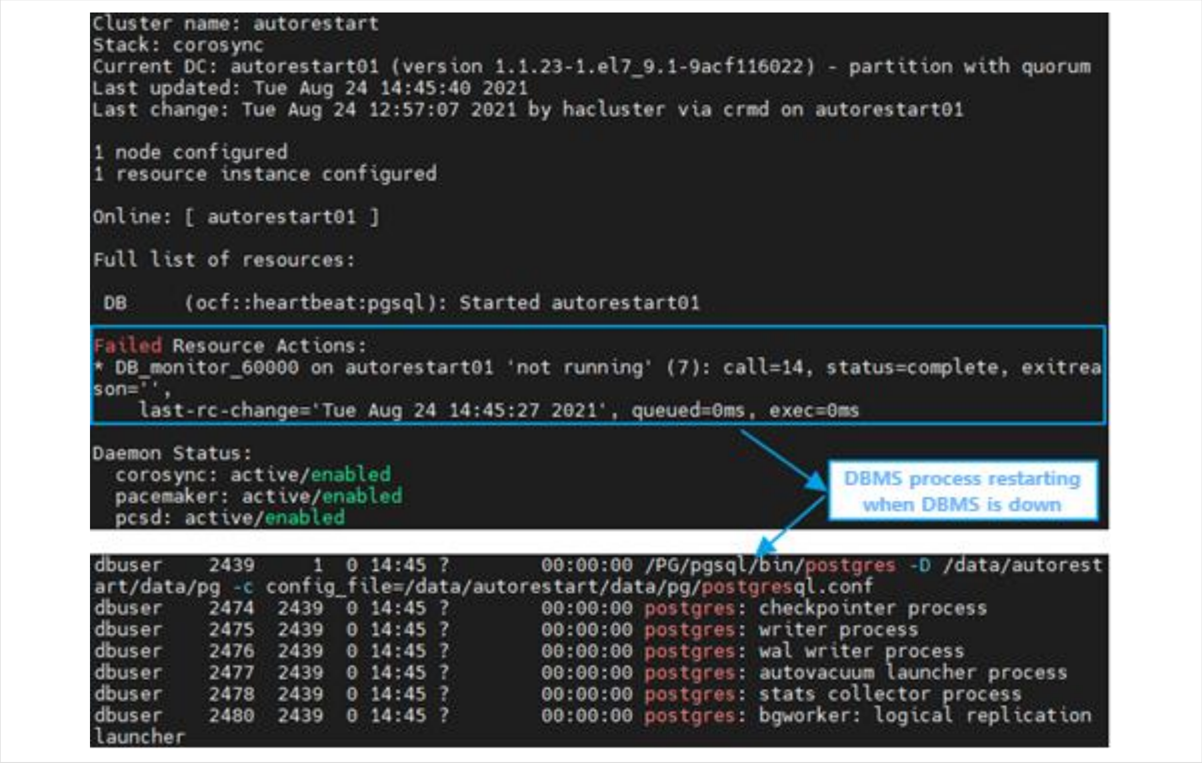
The following is a configuration example in which DB resources are registered with a single node cluster using Pacemaker.



```
Cluster name: autorestart
Stack: corosync
Current DC: autorestart01 (version 1.1.23-1.el7_9.1-9acf116022) - partition with quorum
Last updated: Tue Aug 24 12:57:31 2021
Last change: Tue Aug 24 12:57:07 2021 by hacluster via crmd on autorestart01

1 node configured
1 resource instance configured

Online: [ autorestart01 ]

Full list of resources:

 DB     (ocf::heartbeat:pgsql): Started autorestart01

Daemon Status:
  corosync: active/enabled
  pacemaker: active/enabled
  pcsd: active/enabled
 Resource: DB (class=ocf provider=heartbeat type=pgsql)
  Attributes: monitor_user=pgsys pgctl=/PG/pgsql/bin/pg_ctl pgdata=/data/autorestart/dat
a/pg pgdba=dbuser pgport=5432 psql=/PG/pgsql/bin/psql
  Operations: demote interval=0s timeout=120s (DB-demote-interval-0s)
              methods interval=0s timeout=5s (DB-methods-interval-0s)
              monitor interval=60 timeout=120 (DB-monitor-interval-60)
              notify interval=0s timeout=90s (DB-notify-interval-0s)
              promote interval=0s timeout=120s (DB-promote-interval-0s)
              start interval=0s timeout=120s (DB-start-interval-0s)
              stop interval=0s timeout=120s (DB-stop-interval-0s)
```

**Figure 5. An example of Pacemaker configuration**

Test the DB resource to make sure it works properly.
- Test Case: Verification of DBMS process restarting when DBMS process is down



```
Cluster name: autorestart
Stack: corosync
Current DC: autorestart01 (version 1.1.23-1.el7_9.1-9acf116022) - partition with quorum
Last updated: Tue Aug 24 14:45:40 2021
Last change: Tue Aug 24 12:57:07 2021 by hacluster via crmd on autorestart01

1 node configured
1 resource instance configured

Online: [ autorestart01 ]

Full list of resources:

 DB     (ocf::heartbeat:pgsql): Started autorestart01

Failed Resource Actions:
* DB_monitor_60000 on autorestart01 'not running' (7): call=14, status=complete, exitrea
son='',
    last-rc-change='Tue Aug 24 14:45:27 2021', queued=0ms, exec=0ms

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

```
dbuser    2439     1  0 14:45 ?        00:00:00 /PG/pgsql/bin/postgres -D /data/autorest
art/data/pg -c config_file=/data/autorestart/data/pg/postgresql.conf
dbuser    2474  2439  0 14:45 ?        00:00:00 postgres: checkpointer process
dbuser    2475  2439  0 14:45 ?        00:00:00 postgres: writer process
dbuser    2476  2439  0 14:45 ?        00:00:00 postgres: wal writer process
dbuser    2477  2439  0 14:45 ?        00:00:00 postgres: autovacuum launcher process
dbuser    2478  2439  0 14:45 ?        00:00:00 postgres: stats collector process
dbuser    2480  2439  0 14:45 ?        00:00:00 postgres: bgworker: logical replication
launcher
```

**Figure 6. Restart test when DBMS process is down**

## 6.1 Considerations

Failover by virtual server HA using hypervisor clustering (more than 10 minutes) takes more time than failover using general application HA consisting of two or more nodes (within 1-2 minutes). In addition, there may be limitations in responding to storage and network failures. Therefore, it is necessary to analyze the level of availability required for each system and select the appropriate high availability configuration accordingly.

To conclude, we have looked at the AutoRestart function of DB Service to increase the availability of a DBMS composed of a single virtual server in a cloud environment.