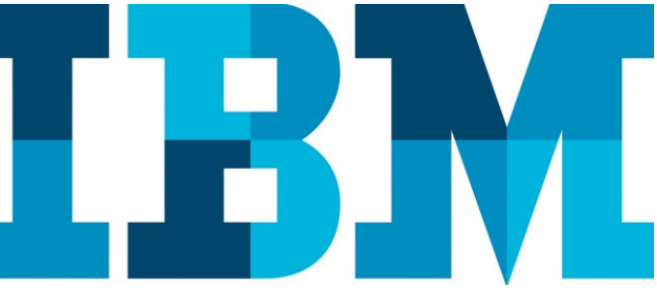
PostgreSQL agent for IBM VM Recovery Manager

Configuring and monitoring PostgreSQL agent through IBM VM Recovery Manager on Linux

Table of contents



Overview

Challenge
Administrators need to manually monitor and make sure that the PostgreSQL applications are always up and running to prevent data loss. They also need to maintain replication between such instances for high availability.

Solution
This paper provides detailed explanation about PostgreSQL application and its services and helps administrators to achieve high availability of such applications using the IBM VM Recovery Manager solution.

Introduction

PostgreSQL is an open source object-relational database system which extends Structured Query Language (SQL) to store and scale complicated data workloads. The origin of PostgreSQL date back to 1986 as part of the POSTGRES project at the University of California at Berkeley and has more than 30 years of active development on the core platform.

This paper provides an overview of PostgreSQL, its installation steps, and how PostgreSQL databases are created for a specific PostgreSQL instance owner and the database interaction with each other and achieving high availability of such applications using the IBM® VM Recovery Manager HA for Power Systems solution.

The intended audience for this paper includes system administrators or any user of the VM Recovery Manager HA solution.

Prerequisites

As a prerequisite, knowledge about the following topics would be helpful:

- PostgreSQL database application
- VM agent
- KSYS sub system

PostgreSQL Database

PostgreSQL is a powerful, open-source, and an object-relational database system that uses and extends the SQL combined with many features that safely store and scale the most complicated data workloads.

PostgreSQL has earned a strong reputation for its proven architecture, reliability, data integrity, robust feature set, extensibility, and the dedication of the open source community behind the software to consistently deliver performant and innovative solutions.

PostgreSQL process types

PostgreSQL has the following four process types:

- Postmaster (Daemon) process
- Background process
- Back-end process
- Client process

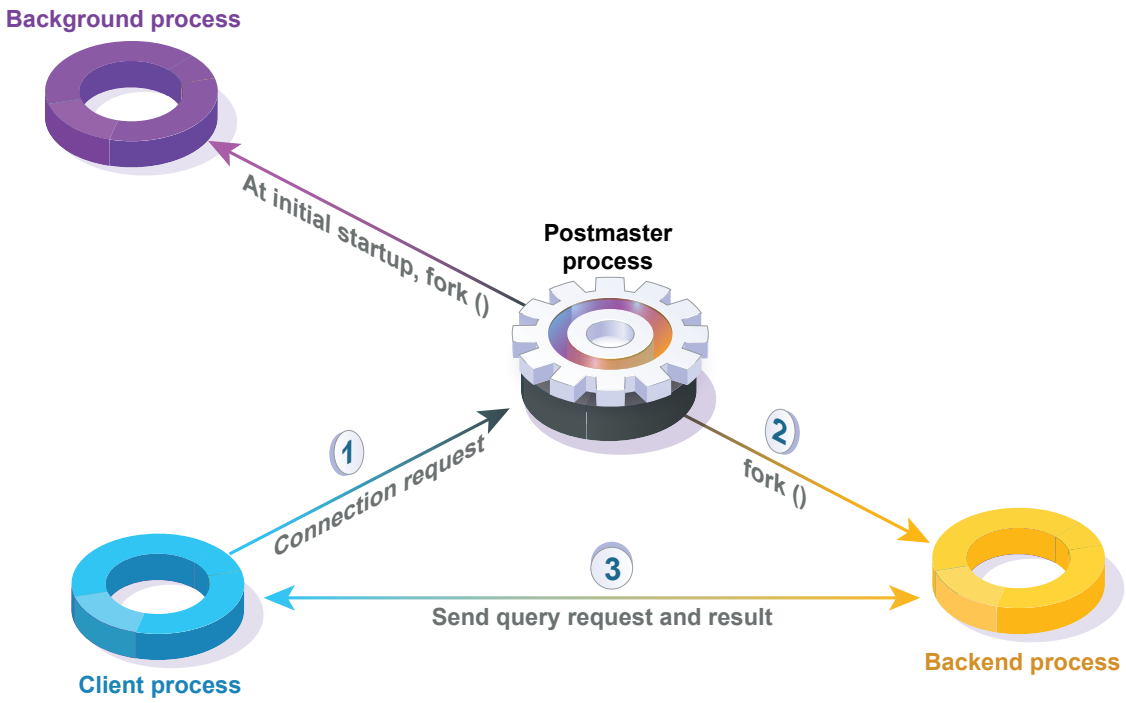


Figure 1. PostgreSQL application process types

Postmaster process

The postmaster process is the first process that is started when you start PostgreSQL. At startup, this process performs recovery, initializes shared memory, and runs background processes. It also creates a backend process when there is a connection request from the client process.

If you check the relationships between processes using the `pstree` command, you can see that the postmaster process is the parent process of all the processes. In the example shown in Figure 2, the process name and argument are added after the process ID.

```
$ pstree -p 1125
postgres(1125) /usr/local/pgsql/bin/postgresql -D /usr/local/pgsql/data
        ____ postgres(1249) postgres: logger process
        |__ postgres(1478) postgres: checkpointer process
        |__ postgres(1479) postgres: writer process
        |__ postgres(1480) postgres: wal writer process
        |__ postgres(1481) postgres: autovacuum launcher process
        |__ postgres(1482) postgres: archiver process
        |__ postgres(1483) postgres: stats collector process
```

Figure 2. Checking the postgres process and its worker process

Background process

The following table shows the list of background processes required for the PostgreSQL operation.

Process	Role
Logger	Writes the error message to the log file.
Checkpointer	Writes the dirty buffer to a file when a checkpoint occurs.
Writer	Writes the dirty buffer to a file periodically.
Wal writer	Writes the WAL buffer to the WAL file.
Autovacuum launcher	Forks autovacuum worker when autovacuum is enabled. It is the responsibility of the autovacuum daemon to carry vacuum operations on bloated tables on demand.
Archiver	Copies the WAL file to the specified directory when in the Archive.log mode.
Stats collector	Collects DBMS usage statistics such as session execution information (`pg_stat_activity`) and table usage statistical information (`pg_stat_all_tables`).

Table 1. PostgreSQL background processes

Back-end process

The maximum number of back-end processes is set by the `max_connections` parameter, and the default value is `100`. The backend process performs the query request of the user process and then transmits the result. Some memory structures are required for query execution, which is called local memory. The main parameters associated with local memory are:

- `work_mem`
  Space used for sorting, bitmap operations, hash joins, and merge joins. The default setting is 4 MB.
- `maintenance_work_mem`
  Space used for vacuum and CREATE INDEX. The default setting is 64 MB.
- `temp_buffers`
  Space used for temporary tables. The default setting is 8 MB.

Client process

Client process refers to the background process that is assigned for every back-end user connection. Usually, the postmaster process will fork a child process that is dedicated to serve a user connection.

PostgreSQL is implemented using a simple *process per user* client/server model. In this model, there is one client process that is connected to exactly one server process. As you do not know ahead of time how many connections will be made, you have to use a master process that spawns a new server process every time a connection is requested. This master process is called `postgres` and listens at a specified TCP/IP port for incoming connections. Whenever a request for a connection is detected, the `postgres` process spawns a new server process. The server tasks communicate with each other using semaphores and shared memory to ensure data integrity throughout concurrent data access.

After a connection is established, the `client process` can send a query to the back end (server). The query is transmitted using plain text, that is, there is no parsing done in the front-end (client). The server parses the query, creates an execution plan, executes the plan, and returns the retrieved rows to the client by transmitting them over the established connection.

Architecture

Software

- PostgreSQL on Linux (RHEL/SUSE)

Hardware

- 4-core CPU
- 2 GB RAM
- 40 GB hard disk space

Architecture overview

The following figure provides the architecture overview of the VM Recovery Manager solution and PostgreSQL application

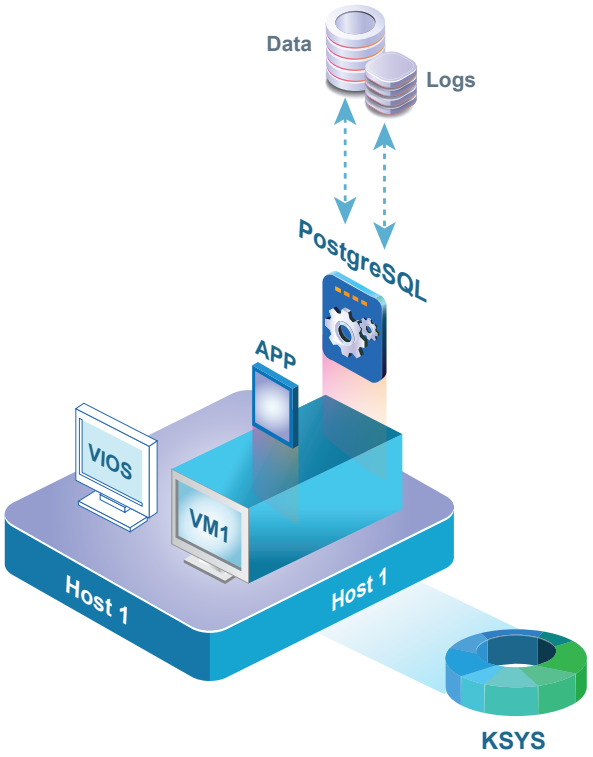


Figure 3. PostgreSQL application installed on VMs with KSYS and VIOS

The following list explains the components in the figure.

- **VM**: Virtual machines (VMs), also known as the logical partitions (LPARs).
- **HA**: High availability, to provide continuous processing for all important applications.
- **KSYS**: The controlling system, which is a controlling software for the high availability (HA) operation.
- **VIOS**: A special LPAR that hosts I/O resources to provide advanced virtualization capabilities across other client logical partitions (or VMs).
- **PostgreSQL**: PostgreSQL is a powerful open-source, and an object-relational database system.
- **VM Agent**: VM Monitor daemon, which is used to monitor the virtual machine and to monitor applications on the virtual machines.

Installation

This section explains how to install the required files for PostgreSQL application on the VM using the `rpm` command.

Download the following *rpm* files from the downloads page and select the respective Linux distribution:

- postgresql-libs-x.x.xx
- postgresql-x.x.xx
- postgresql-server-x.x.xx

Then, install PostgreSQL on the Linux VM.

```
(0) root @ rt18004: /
# rpm -ivh postgresql-libs-9.2.21-1.el7.ppc64le.rpm
warning: postgresql-libs-9.2.21-1.el7.ppc64le.rpm: Header V3 RSA/SHA256 Signature
Preparing...                          ################################# [100%]
Updating / installing...
1:postgresql-libs-9.2.21-1.el7        ################################# [100%]

(0) root @ rt18004: /
# rpm -ivh postgresql-9.2.21-1.el7.ppc64le.rpm
warning: postgresql-9.2.21-1.el7.ppc64le.rpm: Header V3 RSA/SHA256 Signature
Preparing...                          ################################# [100%]
Updating / installing...
1:postgresql-9.2.21-1.el7             ################################# [100%]

(0) root @ rt18004: /
# rpm -ivh postgresql-server-9.2.21-1.el7.ppc64le.rpm
warning: postgresql-server-9.2.21-1.el7.ppc64le.rpm: Header V3 RSA/SHA256 Signature
Preparing...                          ################################# [100%]
Updating / installing...
1:postgresql-server-9.2.21-1.el7      ################################# [100%]
```

Figure 4. PostgreSQL rpm installation

Start/Stop PostgreSQL service

After the PostgreSQL installation is completed, the user can verify the PostgreSQL service using the following command:

```
service postgresql.service status
```

Figure 5 shows the output of the `postgresql service status` command.

```
(1) root @ rt18004: /
# service postgresql status;
● postgresql.service - PostgreSQL database server
   Loaded: loaded (/usr/lib/systemd/system/postgresql.service; disabled;
           vendor preset: disabled)
   Active: inactive (dead)
```

Figure 5. PostgreSQL service status output as inactive

And the user can also stop/start the PostgreSQL service using the following commands:

```
service postgresql.service stop
```

```
service postgresql.service start
```

```
(0) root @ rt18004: /
# service postgresql status;
● postgresql.service - PostgreSQL database server
   Loaded: loaded (/usr/lib/systemd/system/postgresql.service; disabled;
           vendor preset: disabled)
   Active: active (running) since Fri 2019-11-29 01:25:04 EST; 3s ago
  Process: 4212 ExecStart=/usr/lib/postgresql-init start
           (code=exited, status=0/SUCCESS)
 Main PID: 4223 (postgres)
    Tasks: 7 (limit: 512)
   CGroup: /system.slice/postgresql.service
           ├─4223 /usr/lib/postgresql96/bin/postgres -D /var/lib/pgsql/data
           ├─4224 postgres: logger process
           ├─4226 postgres: checkpointer process
           ├─4227 postgres: writer process
           ├─4228 postgres: wal writer process
           ├─4229 postgres: autovacuum launcher process
           └─4230 postgres: stats collector process
Nov 29 01:25:03 rt18004 systemd[1]: Starting PostgreSQL database server...
Nov 29 01:25:04 rt18004 postgresql-init[4212]: Log will be in directory "pg_log".
Nov 29 01:25:04 rt18004 systemd[1]: Started PostgreSQL database server.
```

Figure 6. PostgreSQL service status output as active

Create PostgreSQL database directory

This section explains a few important points that can help in understanding the database directory structure of PostgreSQL.

Components related to the database:

- PostgreSQL consists of several databases. This is called a database cluster.
- When the `initdb()` command is run, the data storage area is initialized and thus `template0`, `template1`, and `postgres` databases are created.
- The `template0` and `template1` databases are template databases for user database creation and contain the system catalog tables.
- The list of tables in the template0 and template1 databases is the same immediately after `initdb()`. Whenever a new database is created within the cluster, template1 is essentially cloned. This means that any
  changes you make in template1 are propagated to all subsequently created databases. The template0 database contains the same data as template1, that is, only the standard objects predefined by your version of PostgreSQL. The template0 database should never be changed after the database cluster has been initialized. However, the template1 database can create objects that the user needs.
- The user database is created by cloning the template1 database.

Components related to the tablespace:

- The pg_default and pg_global tablespaces are created immediately after initdb().
- If you do not specify a tablespace at the time of table creation, it is stored in the pg_dafault tablespace.
- Tables managed at the database cluster level are stored in the pg_global tablespace.
- The physical location of the pg_default tablespace is $PGDATA\base.
- The physical location of the pg_global tablespace is $PGDATA\global.
- One tablespace can be used by multiple databases.In that case, a database-specific subdirectory is created in the tablespace directory.

You must create the database directory for the postgresql daemon, where the PostgreSQL database along with template0 and template1 databases will be created by the postgresql daemon.

1. Create the database directory in the file system.

```
(0) root @ rt18004: /
# date;mkdir /home/database_directory/
Tue Sep 29 06:00:44 EDT 2020
(0) root @ rt18004: /
ls -ltr /home
drwxr-x---   2   root   root    4096  Oct  6  01:02   apps
drwxr-xr-x   2   root   root       6  Dec 22  02:23   database_directory
```

Figure 7. PostgreSQL database directory

2. Change the ownership for that database directory and assign to the postgres user.

```
(0) root @ rt18004: /
# date;chown -R postgres /home/database_directory/
Tue Sep 29 06:00:44 EDT 2020
(0) root @ rt18004: /
ls -ltr /home
drwxr-x---   2   root       root    4096  Oct  6  01:02   apps
drwxr-xr-x   2   postgres   root       6  Dec 22  02:23   database_directory
```

Figure 8. Changing the PostgreSQL database directory user ownership

3. Initialize the database in the directory created by logging in as the postgres user using the following command:
   `pg_ctl -D /home/database_directory/database init`

```
(0) root @ rt18004: /
#  su - postgres
Last login: Tue Sep 29 06:00:25 EDT 2020 on hvc0
-bash-4.2$
-bash-4.2$ pg_ctl -D /home/database_directory/database init
The files belonging to this database system will be owned by user "postgres".
This user must also own the server process.
The database cluster will be initialized with locale "en_US.UTF-8".
The default database encoding has accordingly been set to "UTF8".
The default text search configuration will be set to "english".

creating directory /home/database_directory/database ... ok
creating subdirectories ... ok
selecting default max_connections ... 100
selecting default shared_buffers ... 32MB
creating configuration files ... ok
creating template1 database in /home/database_directory/database/base/1 ... ok
initializing pg_authid ... ok
initializing dependencies ... ok
creating system views ... ok
loading system objects descriptions ... ok
creating collations ... ok
creating conversions ... ok
creating dictionaries ... ok
setting privileges on built-in objects ... ok
creating information schema ... ok
loading PL/pgSQL server-side language ... ok
vacuuming database template1 ... ok
copying template1 to template0 ... ok
copying template1 to postgres ... ok
WARNING: enabling "trust" authentication for local connections
You can change this by editing pg_hba.conf or using the option -A, or
--auth-local and --auth-host, the next time you run initdb.
Success. You can now start the database server using:
    /usr/bin/postgres -D /home/database_directory/database
or
    /usr/bin/pg_ctl -D /home/database_directory/database -l logfile start
```

Figure 9. Initializing the PostgreSQL database

4. Start the database created by assigning the logfile.

```
-bash-4.2$ /usr/bin/pg_ctl -D /home/database_directory/database -l logfile start
server starting...
-bash-4.2$ /usr/bin/pg_ctl -D /home/database_directory/database -l logfile status
pg_ctl: server is running (PID: 21440)
/usr/bin/postgres "-D" "/home/database_directory/database"
```

Figure 10. Starting the PostgreSQL database with logfile

5. Log in to the database.

```
-bash-4.2$ psql
psql (9.2.21)
Type "help" for help.

postgres=# \l
                                  List of databases
   Name    |  Owner   | Encoding |   Collate   |    Ctype    |   Access privileges
-----------+----------+----------+-------------+-------------+-----------------------
 postgres  | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 template0 | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/postgres          +
           |          |          |             |             | postgres=CTc/postgres
 template1 | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/postgres          +
           |          |          |             |             | postgres=CTc/postgres
(3 rows)
postgres=#
```

Figure 11. Database login

6. Check the database server process status and verify if the PostgreSQL database server process is running.

```
  (0) root @ rt18004: /
  # ps -ef | grep postgres
  root     2303       1 0  Dec 08  ? 00:00:01  /usr/libexec/postfix/master -w
  postfix  2305    2303 0  Dec 08  ? 00:00:00  qmgr -l -t unix -u
  postfix  26999   2303 0   02:00  ? 00:00:00  pickup -l -t unix -u
  postgres 29264      1 0   02:28  ? 00:00:00  /usr/bin/postgres -
D /home/database_directory/database
  postgres 29265  29264 0   02:28  ? 00:00:00  postgres: logger process
  postgres 29267  29264 0   02:28  ? 00:00:00  postgres: checkpointer process
  postgres 29268  29264 0   02:28  ? 00:00:00  postgres: writer process
  postgres 29269  29264 0   02:28  ? 00:00:00  postgres: wal writer process
  postgres 29270  29264 0   02:28  ? 00:00:00  postgres: autovacuum launcher process
  postgres 29271  29264 0   02:28  ? 00:00:00  postgres: stats collector process
  root     29271  25183 0   02:28  ? 00:00:00  grep --color=auto post
```

Figure 12. Checking the PostgreSQL process

Integration with VM Recovery Manager HA

This section explains the procedure to integrate the PostgreSQL agent with the VM Recovery Manager.

Prerequisites:

- One VM with required hardware with PostgreSQL database configured and running.
- A KSYS subsystem managing the VM.

Configuring PostgreSQL with default scripts

If you configure the high availability function at the VM level or the application level, you must set up the VM agent.

Agent scripts

The KSYS VM daemon uses the following agent scripts to start, stop, and monitor the PostgreSQL application:

- `/usr/sbin/agents/postgres/startpostgres`
- `/usr/sbin/agents/postgres/stoppostgres`
- `/usr/sbin/agents/postgres/monitorpostgres`

The ksysvmmgr command provides a consistent interface for the VM agent to manage the virtual machine and the applications that are running in the virtual machine.

Run the following command to add the PostgreSQL VM agent:
ksysvmmgr [-s] [-l {0|1|2|3}] add app <NAME> type=POSTGRES instancename=<VALUE#1> database=<VALUE#2> configfile=<VALUE#3> [<ATTR#n>=<VALUE#n>]

Example:

Add the PostgreSQL application:

```
Ksysvmmgr -s add application postgresqlapp instancename=postgres
database=testdb type=POSTGRES config
file=/var/ksys/config/samples/POSTGRESconfig.xml
```

In this example, **postgres** is the database username, and **testdb** is the database name.

The following table explains the attributes used in the ksysvmmgr command.

Attribute	Description
type	White creating a PostgreSQL application, the type must have the value POSTGRES.
instancename	The instancename attribute must be specified with the PostgreSQL username
Database	The database attribute must be specified with the PostgreSQL database name.
configfile	The configfile attribute specifies the file path of the configuration file, which stores the settings of the application configuration.

Table 2. ksysvmmgr command attributes

Users must specify the path of the configuration file while adding the PostgreSQL application. A sample configuration file is POSTGRESconfig.xml, provided in the /var/ksys/config/samples folder. You can use this sample file by updating the attribute values. If you do not specify the configuration file path or appropriate values in the configuration file, the PostgreSQL application will be not be added.

The POSTGRESconfig.xml file contains the following attributes:

POSTGRESinstance id and data_directory

The following table describes the attributes present in the PostgreSQL configuration file.

Attribute name	Description
POSTGRESinstance id	The POSTGRESinstance id attribute is the PostgreSQL instance name
data_directory	It specifies the directory where the database is created.

Table 3. PostgreSQL application configuration file attributes with pre-defined scripts

Refer to an example PostgreSQL configuration file without replication setup:

```
<?xml version="1.0" encoding="UTF-8"?>
<POSTGRESConfig>
  <!--NOTE: PLEASE UNCOMMENT THE REQUIRED ATTRS AND SET APPROPRIATE VALUES.-
->
  <POSTGRESinstance id="postgres">
    <data_directory>/var/lib/pgsql/data</data_directory>
  </POSTGRESinstance>
  <!--POSTGRESinstance id="instOwner"-->
    <!--data_directory>/var/lib/pgsql/data</data_directory-->
  <!--/POSTGRESinstance-->
</POSTGRESConfig>
```

After adding the PoatgreSQL agent, using the `ksysmgr query app` command, users can query the application and validate the attributes.

Configuring PostgreSQL with user-specified scripts

The user can also provide the custom start, stop, and monitor scripts to the KSYS VM daemon to add the PostgreSQL application.

```
ksysvmmgr [-s] [-l {0|1|2|3}] add app <NAME> type=POSTGRES
start_script=<VALUE#1> stop_script=<VALUE#2> monitor_script=<VALUE#3>
instancename=<VALUE#4> database=<VALUE#5> configfile=<VALUE#6>
[<ATTR#n>=<VALUE#n>]
```

Example:

```
ksysvmmgr add application postgresqlapp
start_script=/home/postgresql_start_script
stop_script=/home/postgresql_stop_script
monitor_script=/home/postgresql_monitor_scriptinstancename=postgres
database=database type=POSTGRES config
file=/var/ksys/config/samples/POSTGRESconfig.xml
```

The following table describes the PostgreSQL application configuration file attributes:

Attribute	Description
`type`	White creating a PostgreSQL application, the `type` must have the value POSTGRES.
`instancename`	The `instancename` must attribute must be specified with PostgreSQL username
`database`	The `database` attribute must be specified with the PostgreSQL database name.
`configfile`	The `configfile` attribute specifies the file path of the configuration file, which stores the settings of the application configuration.
`start_script`	The attribute specifies the location of the start script of the PostgreSQL application.
`stop_script`	The attribute specifies the location of the stop script of the PostgreSQL application
`monitor_script`	The attribute specifies the location of the monitor script of the PostgreSQL application

Table 4. PostgreSQL application configuration file attributes with user-defined scripts

Summary

This white paper enables users to understand the end-to-end HA solution by IBM VM Recovery Manager and the process involved in configuring and monitoring the PostgreSQL application and its health, and auto-restart applications whenever required. To ensure that the correct functionality is used, it is always recommended to consider the prerequisites and limitations before installing and running PostgreSQL with a VM agent in VM Recovery Manager.

Related links

For a basic understanding of the HA mechanism and application management engine process check the following references:

- VM Agents
- IBM VM Recovery Manager for IBM Power Systems
- Configuring PostgreSQL high availability
- PostgreSQL Documentation

About the author

Adari Naga Arvind is an advisory software engineer in the VM Recovery Manager product team. He has more than 2 years of experience in the IBM Power® platform and has knowledge on disaster recovery and high availability. You can reach Arvind at arvadari@in.ibm.com.

IBM Systems
3039 Cornwallis Road
RTP, NC 27709

Produced in the United States of America