HIE 3.PostgreSQL Installation Document

Table of Contents

[1 About this document 2](#_Toc59439846)

[1.1 Purpose of this document 2](#_Toc59439847)

[1.2 Audience 2](#_Toc59439848)

[1.3 Abbreviations 2](#_Toc59439849)

[1.4 Assumptions 2](#_Toc59439850)

[1.5 Open items 2](#_Toc59439851)

[2 Overview 3](#_Toc59439852)

[3 Install postgresql-12 on centos-7 4](#_Toc59439853)

[3.1.1 PostgreSQL 12 on CentOS 7 4](#_Toc59439854)

[3.1.2 Initialize and Start Database service 4](#_Toc59439855)

[4 postgresql-12 configuration 6](#_Toc59439856)

[4.1 Set PostgreSQL admin user’s password 6](#_Toc59439857)

[4.2 Enable remote access 6](#_Toc59439858)

[5 replication set-up 8](#_Toc59439859)

[5.1 Setting Up the Master PostgreSQL Server 8](#_Toc59439860)

[5.2 Configure slave – Server 9](#_Toc59439861)

[6 changed parameters 11](#_Toc59439862)

# About this document

## Purpose of this document

The purpose of this document is to guide on how to install the PostgreSQL-12 on centos 7.

## Audience

All the Devops and network administrator level users of MDAP are the audience of this document.

## Abbreviations

|  |  |
| --- | --- |
| **Abbreviation** | **Expansion** |
| WAL | Write Ahead Log |

## Assumptions

The Assumptions that are made in the document are

## Open items

List of open items.

# Overview

PostgreSQL is an object-relational database management system. It is a highly stable database management system, backed by more than 20 years of community development which has contributed to its high levels of resilience, integrity, and correctness. PostgreSQL is used as the primary data store or data warehouse for many web, mobile, geospatial, and analytics applications

# Install postgresql-12 on centos-7

* The PostgreSQL Yum Repository will integrate with your normal systems and patch management and provide automatic updates for all supported versions of PostgreSQL throughout the support lifetime of PostgreSQL.

yum -y install https://download.postgresql.org/pub/repos/yum/reporpms/EL-7-x86\_64/pgdg-redhat-repo-latest.noarch.rpm

You can get more information on installed package by running the command:

rpm -qi pgdg-redhat-repo

### PostgreSQL 12 on CentOS 7

Install PostgreSQL client and server packages:

sudo yum -y install epel-release yum-utils

sudo yum-config-manager --enable pgdg12

sudo yum install postgresql12-server postgresql12 postgresql12-contrib

### Initialize and Start Database service

After installation, database initialization is required before service can be started.

sudo /usr/pgsql-12/bin/postgresql-12-setup initdb

Start and enable the database server service.

sudo systemctl enable --now postgresql-12

Confirm that the service is started without any errors.

systemctl status postgresql-12

If you have a running Firewall service and remote clients should connect to your database server, allow PostgreSQL service.

sudo firewall-cmd --add-service=postgresql --permanent

sudo firewall-cmd --reload

# postgresql-12 configuration

For accessing the remote connection, we need to configure default postgresql configurations.

## Set PostgreSQL admin user’s password

Set PostgreSQL admin user

sudo su - postgres

psql -c "alter user postgres with password 'StrongPassword'"

ALTER ROLE

## Enable remote access

Edit the file /var/lib/pgsql/12/data/postgresql.conf and set Listen address to your server IP address or “**\***” for all interfaces.

listen\_addresses = '192.168.10.10'

Also set PostgreSQL to accept remote connections

$ sudo vim /var/lib/pgsql/12/data/pg\_hba.conf

# Accept from anywhere

host all all 0.0.0.0/0 md5

# Accept from trusted subnet

host all all 192.168.1.0/24md5

Restart database service after committing the change.

sudo systemctl restart postgresql-12

Connecting to remote database:

psql -U <dbuser> -h <serverip> -p 5432 <dbname>

# replication set-up

The database records of the **Master** server are duplicated to the **Slave** servers. You can read from the **Slave** servers using the IP addresses of the **Slave** servers. But you add new data only to the **Master** server. The servers are all synced up. So, if any of the **Master** server fails, one of the Slave servers can take over and become a new Master. That is how PostgreSQL can handle database request without any interruption even if some of the servers fail in a **Master**/**Slave** configuration.

## Setting Up the Master PostgreSQL Server

Now create a new user **replication**:

 psql -c "CREATE USER replication REPLICATION LOGIN CONNECTION LIMIT 1 ENCRYPTED  
PASSWORD 'YOUR\_PASSWORD';"

Now open /etc/postgresql/12/data/pg\_hba.conf with vi:

vi /var/lib/postgresql/12/data/pg\_hba.conf

Add the following line to the marked location:

host    replication     replication   192.168.1.0/24  md5

Now open the main PostgreSQL configuration file

vi  /var/lib/postgresql/12/data/postgresql.conf

Now find and change the following settings. If any line is commented out, uncomment it (removing #) as necessary.

listen\_addresses = 'Private\_IP\_ADDRESS'  
wal\_level = replica  
max\_wal\_senders = 10  
wal\_keep\_segments = 64

**Now restart PostgreSQL server on your**pg-master **server:**

systemctl restart postgresql

## Configure slave – Server

Now open /etc/postgresql/12/data/pg\_hba.conf with vi:

vi /var/lib/postgresql/12/data/pg\_hba.conf

Add the following line to the marked location:

host    replication     replication   192.168.1.0/24  md5

Now open the main PostgreSQL configuration file

vi  /var/lib/postgresql/12/data/postgresql.conf

Now open the main PostgreSQL configuration file

vi  /var/lib/postgresql/12/data/postgresql.conf

Now find and change the following settings. If any line is commented out, uncomment it (removing #) as necessary.

listen\_addresses = 'Private\_IP\_ADDRESS'  
wal\_level = replica  
max\_wal\_senders = 10  
wal\_keep\_segments = 64

**Now restart PostgreSQL server on your**pg-master **server:**

systemctl restart postgresql

Now go to your data\_directory:

cd /var/lib/postgresql/12/data

Backup everything from that directory:

mv /data /data-back-up

Now copy the data from the **pg-master** server to the **pg-slave** server’s **data\_directory**:

pg\_basebackup -h master-ip -D /var/lib/postgresql/12/data/ -P -U  
replication --wal-method=fetch

Type in the password for the **postgres** user of the **pg-master** server and press **<Enter>**.

Now create a recovery.conf file in the data\_directory with nano:

vi recovery.conf

Now add the following line to it:

standby\_mode          = 'on'  
primary\_conninfo      = 'host=master-ip port=5432 user=replication password=123'  
trigger\_file = '/tmp/MasterNow'

Start the PostgreSQL Slave server:

systemctl start postgresql

# changed parameters

* **listen\_addresses**

|  |  |
| --- | --- |
| Default | Changed value |
| 'localhost' | IP\_ADDEESS |

It ensures the postgresql server what IP address(es) to listen on.

* **max\_connections**

|  |  |
| --- | --- |
| Default | Changed value |
| 100 | 10000 |

max\_connections determine the maximum number of concurrent connections to the database server.

* **shared\_buffers**

|  |  |
| --- | --- |
| Default | Changed value |
| 128MB | 32GB |

PostgreSQL uses 'double buffering', meaning that PostgreSQL uses its own internal buffer as well as kernel buffered IO. In short, this means that data is stored in memory twice. The value for shared\_buffers should be roughly 25% of the total system RAM for a dedicated DB server.

* **work\_mem**

|  |  |
| --- | --- |
| Default | Changed value |
| 4MB | 128MB |

It Specifies the amount of memory to be used by internal sort operations and hash tables before writing to temporary disk files.

* **maintenance\_work\_mem**

|  |  |
| --- | --- |
| Default | Changed value |
| 64MB | 2GB |

It Specifies the maximum amount of memory to be used by maintenance operations, such as VACUUM, CREATE INDEX, and ALTER TABLE ADD FOREIGN KEY.

* **effective\_io\_concurrency**

|  |  |
| --- | --- |
| Default | Changed value |
| 1 | 200 |

Number of simultaneous requests that can be handled efficiently by the disk subsystem. Sets the number of concurrent disk I/O operations that PostgreSQL expects can be executed simultaneously. Raising this value will increase the number of I/O operations that any individual PostgreSQL session attempts to initiate in parallel.

* **max\_worker\_processes**

|  |  |
| --- | --- |
| Default | Changed value |
| 8 | 24 |

Sets the maximum number of background processes that the system can support. This parameter can only be set at server start. The default is 8. When running a standby server, you must set this parameter to the same or higher value than on the master server. Otherwise, queries will not be allowed in the standby server. When changing this value, consider also adjusting max\_parallel\_workers, max\_parallel\_maintenance\_workers, and max\_parallel\_workers\_per\_gather.

* **max\_parallel\_workers\_per\_gather**

|  |  |
| --- | --- |
| Default | Changed value |
| 2 | 12 |

max\_parallel\_workers\_per\_gather: the number of workers that can assist a sequential scan of a table.

* **max\_parallel\_workers**

|  |  |
| --- | --- |
| Default | Changed value |
| 8 | 24 |

Sets the maximum number of workers that the system can support for parallel queries. The default value is 8. When increasing or decreasing this value, consider also adjusting max\_parallel\_workers\_per\_gather. Also, note that a setting for this value which is higher than max\_worker\_processes will have no effect, since parallel workers are taken from the pool of worker processes established by that setting.

* **wal\_level**

|  |  |
| --- | --- |
| Default | Changed value |
| #replica | replica |

It is used for Master -slave replication mode

* **checkpoint\_timeout**

|  |  |
| --- | --- |
| Default | Changed value |
| 5min | 6min |

Sets the maximum time between automatic WAL checkpoints Maximum time between automatic WAL checkpoints. If this value is specified without units, it is taken as seconds. The valid range is between 30 seconds and one day. The default is five minutes (5min). Increasing this parameter can increase the amount of time needed for crash recovery.

* **max\_wal\_size**

|  |  |
| --- | --- |
| Default | Changed value |
| 1GB | 10GB |

Sets the WAL size that triggers a checkpoint Maximum size to let the WAL grow to between automatic WAL checkpoints. This is a soft limit; WAL size can exceed max\_wal\_size under special circumstances, such as heavy load, a failing archive command, or a high wal\_keep\_segments setting. If this value is specified without units, it is taken as megabytes. The default is 1 GB. Increasing this parameter can increase the amount of time needed for crash recovery.

* **min\_wal\_size**

|  |  |
| --- | --- |
| Default | Changed value |
| 80MB | 2GB |

Sets the WAL size that triggers a checkpoint Maximum size to let the WAL grow to between automatic WAL checkpoints. This is a soft limit; WAL size can exceed max\_wal\_size under special circumstances, such as heavy load, a failing archive command, or a high wal\_keep\_segments setting. If this value is specified without units, it is taken as megabytes. The default is 1 GB. Increasing this parameter can increase the amount of time needed for crash recovery.

* **checkpoint\_completion\_target**

|  |  |
| --- | --- |
| Default | Changed value |
| 0.5 | 0.7 |

Time spent flushing dirty buffers during checkpoint, as fraction of checkpoint interval Specifies the target of checkpoint completion, as a fraction of total time between checkpoints.

* **archive\_mode**

|  |  |
| --- | --- |
| Default | Changed value |
| off | on |

Allows archiving of WAL files using archive\_command When archive\_mode is enabled, completed WAL segments are sent to archive storage by setting archive\_command.

* **archive\_command**

|  |  |
| --- | --- |
| Default | Changed value |
| "" | cp %p /LOCATION/STORE/WALS/%f |

The local shell command to execute to archive a completed WAL file segment. Any %p in the string is replaced by the path name of the file to archive, and any %f is replaced by only the file name. (The path name is relative to the working directory of the server, i.e., the cluster's data directory.) Use %% to embed an actual % character in the command. It is important for the command to return a zero-exit status only if it succeeds. For more information see backup-archiving-wal.

* **restore\_command**

|  |  |
| --- | --- |
| Default | Changed value |
| "" | cp %p /LOCATION/STORE/WALS/%f |

Sets the shell command that will retrieve an archived WAL file The local shell command to execute to retrieve an archived segment of the WAL file series. This parameter is required for archive recovery, but optional for streaming replication. Any %f in the string is replaced by the name of the file to retrieve from the archive, and any %p is replaced by the copy destination path name on the server. (The path name is relative to the current working directory, i.e., the cluster's data directory.) Any %r is replaced by the name of the file containing the last valid restart point. That is the earliest file that must be kept allowing a restore to be restartable, so this information can be used to truncate the archive to just the minimum required to support restarting from the current restore. %r is typically only used by warm-standby configurations (see warm-standby). Write %% to embed an actual % character.

* **max\_wal\_senders**

|  |  |
| --- | --- |
| Default | Changed value |
| #10 | 10 |

Specifies the maximum number of concurrent connections from standby servers or streaming base backup clients (i.e., the maximum number of simultaneously running WAL sender processes). The default is 10. The value 0 means replication is disabled. Abrupt streaming client disconnection might leave an orphaned connection slot behind until a timeout is reached, so this parameter should be set slightly higher than the maximum number of expected clients so disconnected clients can immediately reconnect. This parameter can only be set at server start. Also, wal\_level must be set to replica or higher to allow connections from standby servers. When running a standby server, you must set this parameter to the same or higher value than on the master server. Otherwise, queries will not be allowed in the standby server.

* **wal\_keep\_segments**

|  |  |
| --- | --- |
| Default | Changed value |
| 0 | 40 |

Sets the number of WAL files held for standby servers Specifies the minimum number of past log file segments kept in the pg\_wal directory, in case a standby server needs to fetch them for streaming replication. Each segment is normally 16 megabytes. If a standby server connected to the sending server falls behind by more than wal\_keep\_segments segments, the sending server might remove a WAL segment still needed by the standby, in which case the replication connection will be terminated. Downstream connections will also eventually fail as a result. (However, the standby server can recover by fetching the segment from archive, if WAL archiving is in use.)

* **random\_page\_cost**

|  |  |
| --- | --- |
| Default | Changed value |
| 4 | 1.1 |

Sets the planner's estimate of the cost of a nonsequential fetched disk page. Reducing this value relative to seq\_page\_cost will cause the system to prefer index scans; raising it will make index scans look relatively more expensive. You can raise or lower both values together to change the importance of disk I/O costs relative to CPU costs, which are described by the following parameters.

* **effective\_cache\_size**

|  |  |
| --- | --- |
| Default | Changed value |
| 4GB | 96GB |

Sets the planner's assumption about the total size of the data caches Sets the planner's assumption about the effective size of the disk cache that is available to a single query. This is factored into estimates of the cost of using an index; a higher value makes it more likely index scans will be used, a lower value makes it more likely sequential scans will be used. When setting this parameter, you should consider both PostgreSQL's shared buffers and the portion of the kernel's disk cache that will be used for PostgreSQL data files, though some data might exist in both places. Also, consider the expected number of concurrent queries on different tables, since they will have to share the available space. This parameter has no effect on the size of shared memory allocated by PostgreSQL, nor does it reserve kernel disk cache; it is used only for estimation purposes. The system also does not assume data remains in the disk cache between queries. If this value is specified without units, it is taken as blocks, that is BLCKSZ bytes, typically 8kB. The default is 4 gigabytes (4GB). (If BLCKSZ is not 8kB, the default value scales proportionally to it.)