CYBERTEC
POSTGRESQL SERVICES & SUPPORT

CYBERTEC PGEE

INSTALLATION GUIDE FOR REDHAT

Date: 2024-11-09
Publisher: CYBERTEC PGEE team

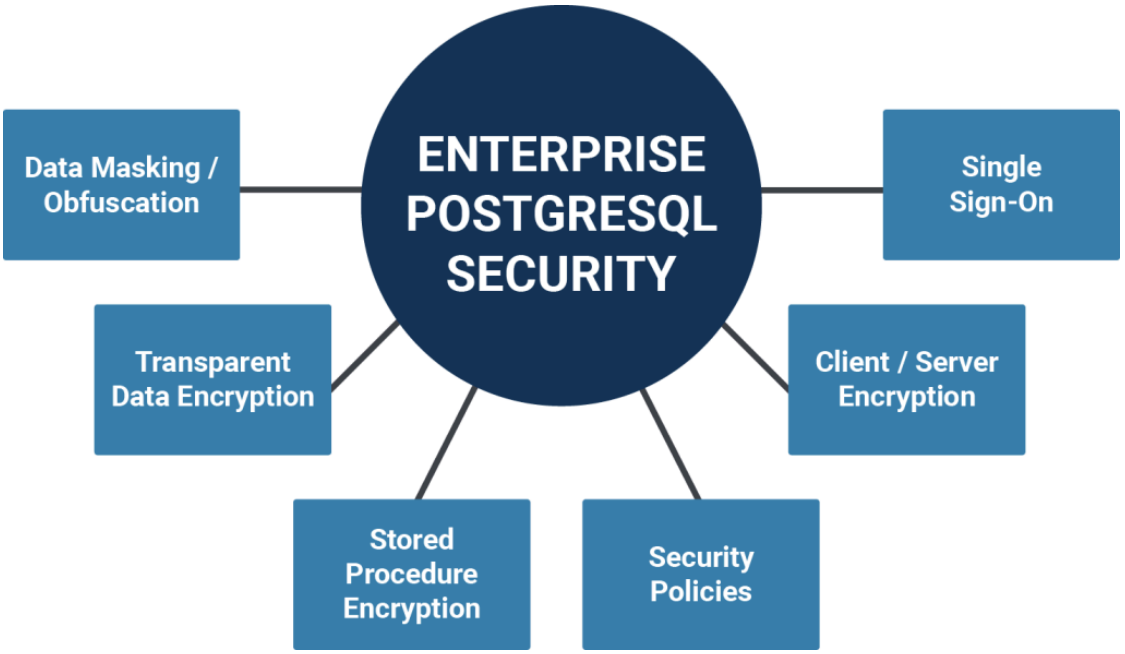
TABLE OF CONTENTS

PGEE: COMPREHENSIVE DATABASE SECURITY

CYBERTEC PostgreSQL Enterprise Edition (PGEE) is a CYBERTEC product which has been designed for enterprise-grade security in critical environments that require additional **security** as well as regular auditing. This solution focus heavily on **compliance** and **business critical** workloads for various industries, including but not limited to:

- Banking and financial services
- Governments and defense
- Critical national infrastructure
- Business-critical missions

Ensuring security is key and therefore our first priority is to provide customers with **encryption at every level** while providing cutting edge performance.



PGEE offers comprehensive database security and provides the necessary tooling to enable enterprise success, focusing on these key aspects:

- Encryption at every level
- Secure software development
- Auditing and certification

This document describes how PGEE can be installed on **RedHat/RPM** based operating systems (RHEL, Rocky Linux, etc.). The following operating systems are currently available and supported:

SUPPORTED PGEE MAJOR VERSIONS AND OPERATING SYSTEMS

- **PGEE 17** based on PostgreSQL 17
- **PGEE 16** based on PostgreSQL 16
- **PGEE 15** based on PostgreSQL 15
- **PGEE 14** based on PostgreSQL 14

- **RedHat RHEL 9** and derivatives, x86_64
- **RedHat RHEL 8** and derivatives, x86_64
- **SUSE SLES 15**, x86_64

Additional operating systems and CPU architectures are supported on request.

INSTALLATION GUIDE

This section contains a detailed step-by-step guide. After the CYBERTEC team has opened the repositories for you, follow the next steps as described in this document:

STEP 1: ACCESS THE RPM REPOSITORY

The RPM repositories can be found here:

https://repository.cybertec.at

Additional instructions can be found in the repository.

STEP 2: DISABLE STANDARD POSTGRESQL

Before installing PGEE we have to disable the onboard PostgreSQL packages to ensure that only the PGEE service is running.

Execute the following command:

```
$ sudo dnf module disable -y postgresql
AlmaLinux 9 - AppStream                  7.5 kB/s | 4.2 kB     00:00
AlmaLinux 9 - AppStream                  9.8 MB/s |  16 MB     00:01
AlmaLinux 9 - BaseOS                     9.4 kB/s | 3.8 kB     00:00
AlmaLinux 9 - BaseOS                     9.9 MB/s |  17 MB     00:01
AlmaLinux 9 - Extras                     8.0 kB/s | 3.3 kB     00:00
AlmaLinux 9 - Extras                      39 kB/s |  20 kB     00:00
Extra Packages for Enterprise Linux 9 - x86_64
                                         247 kB/s |  46 kB     00:00
Extra Packages for Enterprise Linux 9 - x86_64
                                          11 MB/s |  23 MB     00:02
Dependencies resolved.
================================================================
 Package      Architecture    Version   Repository     Size
================================================================
Disabling modules:
 postgresql

Transaction Summary
================================================================
Complete!
```

Once this is done we can proceed with the installation.

STEP 3: SETTING UP THE REPOSITORY

In the next step we have to enable the PGEE repository and make sure we have access to the packages.

Two options are available:

- PGEE demo version, limited to 1 GB per table
- PGEE full version

INSTALLING THE DEMO REPOSITORIES

The demo version can be installed as follows:

```
$ version=16
$ sudo tee /etc/yum.repos.d/cybertec-pg$version.repo <<EOF
[cybertec_pg$version]
name=CYBERTEC PostgreSQL $version for RHEL/CentOS \$releasever - \$basearch
baseurl=https://repository.cybertec.at/public/$version/redhat/\$releasever/\$basearch
gpgkey=https://repository.cybertec.at/assets/cybertec-rpm.asc
enabled=1
EOF
```

ENABLING THE ENTERPRISE REPOSITORIES

As an enterprise customer you will be given access to the full package repository. In this case it is necessary to add the credentials to the repository to the system. The following listing shows how this works:

```
$ version=16 # available: 15 16 17
$ username="YOUR_LOGIN"
$ password="YOUR_PASSWORD"

# RedHat/CentOS
$ sudo tee /etc/yum.repos.d/cybertec-pg$version.repo <<EOF
[cybertec_pg$version]
name=CYBERTEC PostgreSQL $version for RHEL/CentOS \$releasever - \$basearch
baseurl=https://repository.cybertec.at/pgee/$version/redhat/\$releasever/\$basearch
gpgkey=https://repository.cybertec.at/assets/cybertec-rpm.asc
username=$username
password=$password
enabled=1
EOF
```

STEP 4: INSTALL PGEE FROM THE REPOSITORY

Once the repositories have been configured we can move forward and install the desired PGEE packages. In this example we only install the server. However,

```
$ sudo yum install -y postgresql16-ee-server
CYBERTEC PostgreSQL 16 for RHEL/CentOS 9 - x86_64
                                699 kB/s | 226 kB     00:00
Dependencies resolved.
================================================================
 Package                  Arch…   Version                …
================================================================
Installing:
 postgresql16-ee-server  x86_64   16.4-EE~demo.rhel9.1   …
Installing dependencies:
 libicu                  x86_64   67.1-9.el9             …
 lz4                     x86_64   1.9.3-5.el9            …
 postgresql16-ee         x86_64   16.4-EE~demo.rhel9.1   …
 postgresql16-ee-libs    x86_64   16.4-EE~demo.rhel9.1   …
Transaction Summary
================================================================
Install  5 Packages

Total download size: 18 M
Installed size: 72 M
Downloading Packages:
(1/5): lz4-1.9.3-5.el9.x86_64.rpm …
(2/5): postgresql16-ee-libs-16.4-EE~demo.rhel9.1.x86_64.rpm …
(3/5): postgresql16-ee-16.4-EE~demo.rhel9.1.x86_64.rpm …
(4/5): libicu-67.1-9.el9.x86_64.rpm
(5/5): postgresql16-ee-server-16.4-EE~demo.rhel9.1.x86_64.rpm …
----------------------------------------------------------------
Total                             9.6 MB/s |  18 MB    00:01
CYBERTEC PostgreSQL 16 for RHEL/CentOS 9 - x86_64
                                  27 kB/s | 3.1 kB     00:00
Importing GPG key 0x2D1B5F59:
Userid     : "Cybertec International (Software Signing Key)
             <build@cybertec.at>"
 Fingerprint: FCFF 012F 4B39 9019 1352 BB03 AA6F 3CC1 2D1B 5F59
 From       : https://repository.cybertec.at/assets/cybertec-rpm.asc
Key imported successfully
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
  Preparing        :                                              1/1
  Installing       : postgresql16-ee-libs-16.4-EE~demo.rhel9.1.x86_64   1/5
  Running scriptlet: postgresql16-ee-libs-16.4-EE~demo.rhel9.1.x86_64   1/5
  Installing       : libicu-67.1-9.el9.x86_64                     2/5
  Installing       : lz4-1.9.3-5.el9.x86_64                       3/5
  Installing       : postgresql16-ee-16.4-EE~demo.rhel9.1.x86_64  4/5
  Running scriptlet: postgresql16-ee-16.4-EE~demo.rhel9.1.x86_64  4/5
  Running scriptlet: postgresql16-ee-server-16.4-EE~demo.rhel9.1.x86_64  5/5
```

```
  Installing       : postgresql16-ee-server-16.4-EE~demo.rhel9.1.x86_64        5/5
  Running scriptlet: postgresql16-ee-server-16.4-EE~demo.rhel9.1.x86_64        5/5
  Verifying        : libicu-67.1-9.el9.x86_64                                  1/5
  Verifying        : lz4-1.9.3-5.el9.x86_64                                    2/5
  Verifying        : postgresql16-ee-16.4-EE~demo.rhel9.1.x86_64               3/5
  Verifying        : postgresql16-ee-libs-16.4-EE~demo.rhel9.1.x86_64          4/5
  Verifying        : postgresql16-ee-server-16.4-EE~demo.rhel9.1.x86_64        5/5

Installed:
  libicu-67.1-9.el9.x86_64
  lz4-1.9.3-5.el9.x86_64
  postgresql16-ee-16.4-EE~demo.rhel9.1.x86_64
  postgresql16-ee-libs-16.4-EE~demo.rhel9.1.x86_64
  postgresql16-ee-server-16.4-EE~demo.rhel9.1.x86_64

Complete!
```

Congratulations. You are now ready to deploy PGEE on your system.

STEP 5: CREATING SAMPLE KEYS

PGEE is easy to install. One of the core features of PGEE is to provide TDE (= Transparent Data Encryption). This means that two things have to be taken care of:

- During instance creation a key has to be provided
- During startup the same key has to be provided

For a quick and dirty test, we can simply generate keys and use them for PGEE. Here is one way to do it:

```
$ sudo -u postgres -i
$ version=16
$ PATH=/usr/pgsql-$version/bin:$PATH
$ KEY=$(dd if=/dev/random bs=1k count=1 | md5sum - | cut -d ' ' -f 1)
1+0 records in
1+0 records out
1024 bytes (1.0 kB, 1.0 KiB) copied, 7.9108e-05 s, 12.9 MB/s
```

The $KEY variable will contain a key which will be used to handle encryption.

NOTE: Do not use this method in production because the key will be displayed on the screen and be copied verbatim to postgresql.conf. Please make sure you are using the CYBERTEC key management tools described in the next section to handle keys. Use the echo command only for demo purposes and testing !

In a real world deployment make sure you are using the CYBERTEC key management tool which is described later in this document to manage keys safely.

STEP 6: CREATING THE DATABASE INSTANCE

The difference between a normal installation and an encrypted installation is the necessity to tell PGEE how to obtain a key. The -K option allows us to pass a script to the server which handles the key:

```
$ initdb -D /var/lib/pgsql/$version/data -k -K "echo $KEY"
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 "C.utf8".
The default database encoding has accordingly been set to "UTF8".
The default text search configuration will be set to "english".

Data page checksums are enabled.
Data encryption is enabled.

fixing permissions on existing directory /var/lib/pgsql/16/data ... ok
creating subdirectories ... ok
selecting dynamic shared memory implementation ... posix
selecting default max_connections ... 100
selecting default shared_buffers ... 128MB
selecting default time zone ... UTC
creating configuration files ... ok
running bootstrap script ... ok
performing post-bootstrap initialization ... ok
syncing data to disk ... ok

initdb: warning: enabling "trust" authentication for local connections
initdb: hint: 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:

    pg_ctl -D /var/lib/pgsql/16/data -l logfile start
```

Mind the “Data encryption is enabled” line - it reveals that we are on the right path and encryption is working.

STEP 7: STARTING YOUR PGEE INSTANCE

In the next step we can start the server. There are two ways to do it:

STARTING PGEE MANUALLY

Just like any version of PostgreSQL, PGEE can be started manually. The following command shows how this works:

```
$ pg_ctl -D /var/lib/pgsql/$version/data start
waiting for server to start....2024-11-02 14:04:30.393 UTC [311] LOG:
redirecting log output to logging collector
  process
2024-11-02 14:04:30.393 UTC [311] HINT:  Future log output
  will appear in directory "log".
 done
server started
```

Note that $version is a placeholder for your version of PGEE. You can either hardcode your desired version or use the environment variable we have set in the previous section.

STARTING PGEE USING SYSTEMD

In most cases it is more desirable to start PGEE through systemd. The following two commands will enable PGEE and start the server:

```
$ sudo systemctl enable postgresql-16
$ sudo systemctl start postgresql-16
```

Note that PGEE is a drop-in replacement for vanilla PostgreSQL and thus the process does not differ at all from any other service.

STEP 8: VERIFYING ENCRYPTION

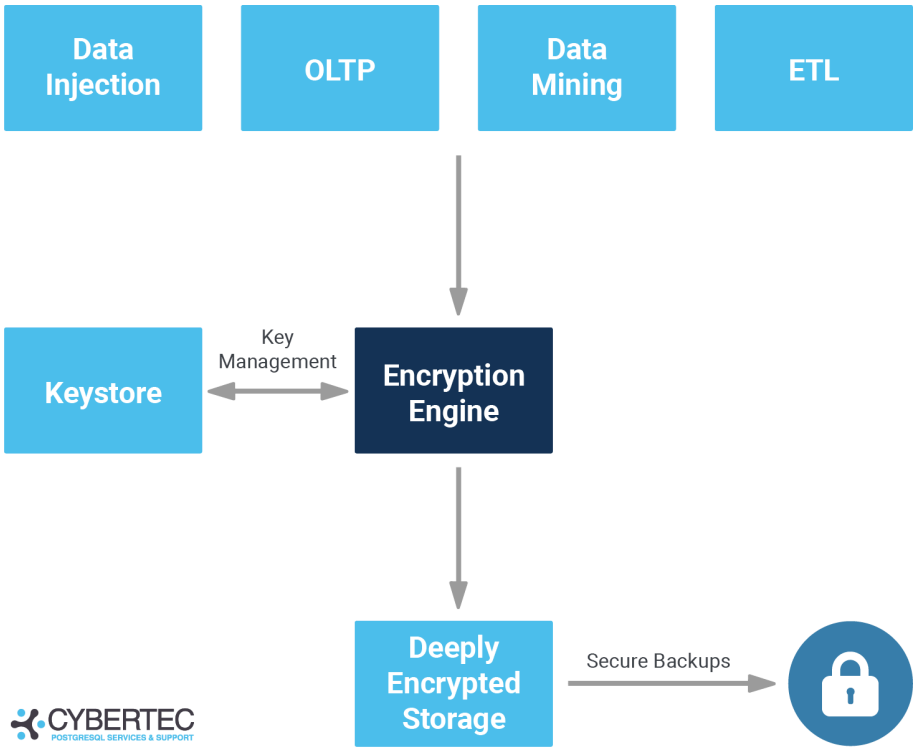
Finally we can verify that encryption is indeed on and working:

```
$ psql
psql (16.4 EE 1.3.7, server 16.4 EE 1.3.7)
Type "help" for help.

postgres=# SHOW data_encryption;
 data_encryption
-----------------
 on
(1 row)
```

MANAGING KEY INTEGRATION

PGEE integrates with every keystore out there. Every time the key is needed it is automatically fetched using a method of your choice which includes but is not limited to:

- Command line prompt (for test purposes only)
- Local files (not recommended, test purposes only)
- Shell output (not recommend)
- CYBERTEC `pgee_key_manager`
  - Recommended solutions
  - Integrates securely with most KSMs
  - Fully supported by CYBERTEC



CYBERTEC PGEE KEY MANAGER

The PGEE key manager avoids keys being leaked. Often database solutions offering Transparent Data Encryption (TDE) allow the key to be leaked on the command line.

This is not so with PGEE. We provide a key management tool which **fetches the key securely** from any location and passes this vital piece of information on to PGEE:

- Without logging the key
- Without exposing the key to the administrators
- Without leaking information
- Without violating security policies

The `pgee_key_manager` works as follows:

- Fetch the key from a location of your choice
- Pass the key safely to PGEE
- Ensure the correct context auf execution
  - Keys can only be obtained in the right content
  - Not key leakages on the command line

> Security information: pgee_key_provider ensures that your key is safely protected. It will only work if called by PGEE directly — otherwise it will simply error out for maximum protection and compliance.

Here is how it works:

```
./pgee_key_manager \
     -command="echo $(openssl rand -hex 16)"
-KeyPath=key.txt
This utility has been executed in the wrong security context.
The incident will be reported.
```

The key manager can be extended and therefore allows for superior flexibility and integration.

UPGRADING FROM POSTGRESQL TO PGEE

If you are already using PostgreSQL (community edition) you can easily **transition to PGEE** without much effort. A handful of steps are needed to make the transition to PGEE and upgrade to the latest version at the same time.

STEP 1: MAKE SURE POSTGRESQL AND PGEE ARE INSTALLED

First of all we have to make sure that PostgreSQL and PGEE are installed. Both packages have to be around to smoothly transition from one database installation to the other.

Note that this is done to ensure that you can upgrade within the same machine without downtime. Execute the following command:

```
# rpm -qa | grep 'postgresql.*server'
postgresql15-server-15.9-1PGDG.rhel9.x86_64
postgresql16-ee-server-16.4-EE.rhel9.1.x86_64
```

In this case we see **PostgreSQL 15** (standard edition) as well as **PGEE 16**. Once this has been verified we can move on to the next step and start the migration process.

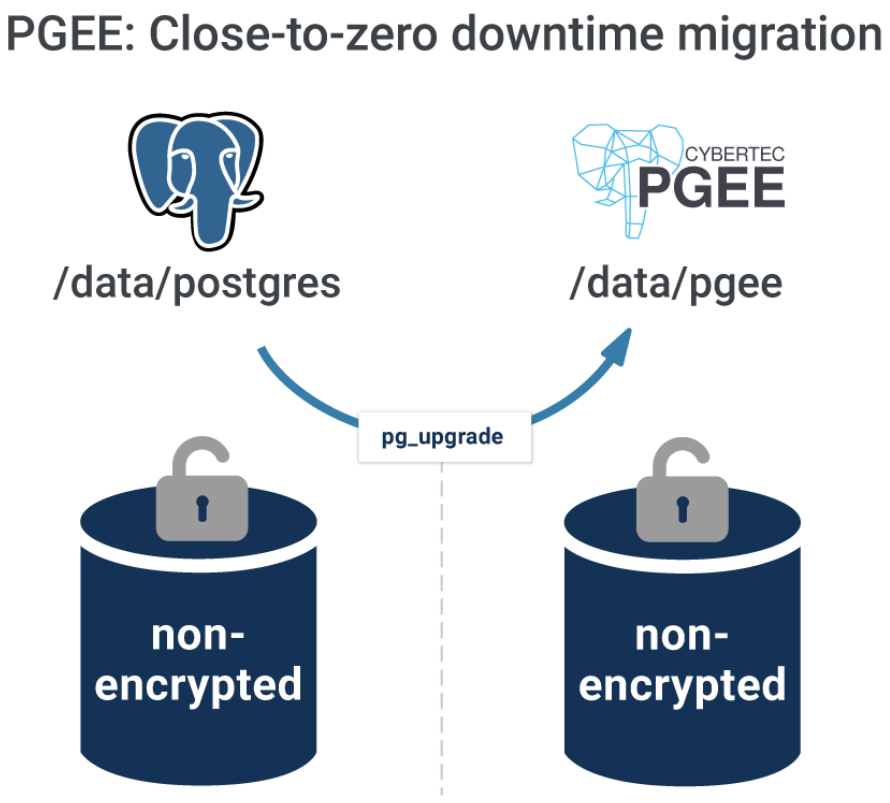
```
# sudo -u postgres /usr/pgsql-16/bin/initdb \
     -D /var/lib/pgsql/16/data/
The files belonging to this database system will be owned by
user "postgres".
This user must also own the server process.
...
```

STEP 2: RUN UPGRADES TO TRANSITION TO PGEE

Verifying your upgrade process is important, so for the sake of simplicity we have created a simple table. The goal is to see this table after our move to PGEE:

```
postgres=# CREATE TABLE documents (doc text);
CREATE TABLE
postgres=# INSERT INTO documents
     VALUES ('My very important document');
INSERT 0 1
```

We use `pg_upgrade` to convert the vanilla cluster (in this example version 15) to PGEE 16.



In this section the process will be explained step by step.

COPYING EXISTING DATA DURING UPGRADES

```
# systemctl stop postgresql-15
# sudo -u postgres -i
$ /usr/pgsql-16/bin/pg_upgrade -b /usr/pgsql-15/bin \
      -d /var/lib/pgsql/15/data \
      -D /var/lib/pgsql/16/data
Performing Consistency Checks
-----------------------------
Checking cluster versions                                   ok
Checking database user is the install user                  ok
Checking database connection settings                       ok
Checking for prepared transactions                          ok
Checking for system-defined composite types in user tables ok
Checking for reg* data types in user tables                 ok
Checking for contrib/isn with bigint-passing mismatch       ok
Checking for incompatible "aclitem" data type in user tables  ok
Creating dump of global objects                             ok
Creating dump of database schemas
                                                            ok
Checking for presence of required libraries                 ok
Checking database user is the install user                  ok
Checking for prepared transactions                          ok
Checking for new cluster tablespace directories             ok

If pg_upgrade fails after this point, you must re-initdb the
new cluster before continuing.

Performing Upgrade
------------------
Setting locale and encoding for new cluster                 ok
Analyzing all rows in the new cluster                       ok
Freezing all rows in the new cluster                        ok
Deleting files from new pg_xact                             ok
Copying old pg_xact to new server                           ok
Setting oldest XID for new cluster                          ok
Setting next transaction ID and epoch for new cluster       ok
Deleting files from new pg_multixact/offsets                ok
Copying old pg_multixact/offsets to new server              ok
Deleting files from new pg_multixact/members                ok
Copying old pg_multixact/members to new server              ok
Setting next multixact ID and offset for new cluster        ok
Resetting WAL archives                                      ok
Setting frozenxid and minmxid counters in new cluster       ok
Restoring global objects in the new cluster                 ok
```

```
Restoring database schemas in the new cluster
                                                        ok
Copying user relation files
                                                        ok
Setting next OID for new cluster                        ok
Sync data directory to disk                             ok
Creating script to delete old cluster                   ok
Checking for extension updates                          ok

Upgrade Complete
----------------
Optimizer statistics are not transferred by pg_upgrade.
Once you start the new server, consider running:
     /usr/pgsql-16/bin/vacuumdb --all --analyze-in-stages
Running this script will delete the old cluster's data files:
     ./delete_old_cluster.sh
```

UPGRADES WITHOUT COPYING THE DATA

The problem with the approach you have just seen is that it will copy all the data. In case of a large database deployment (e.g. many TB) this process takes a lot of time and space. The alternative is to use the link "**-k**" option which creates hard links for the data files used by PGEE.

Here is how it works:

```
# sudo -u postgres -i
$ /usr/pgsql-16/bin/pg_upgrade -b /usr/pgsql-15/bin \
     -d /var/lib/pgsql/15/data \
     -D /var/lib/pgsql/16/data -k
Performing Consistency Checks
-----------------------------
Checking cluster versions                               ok
Checking database user is the install user              ok
Checking database connection settings                   ok
...
Checking for prepared transactions                      ok
Checking for new cluster tablespace directories         ok

If pg_upgrade fails after this point, you must re-initdb the
new cluster before continuing.

Performing Upgrade
```

```
------------------
Setting locale and encoding for new cluster               ok
Analyzing all rows in the new cluster                     ok
...
Restoring global objects in the new cluster               ok
Restoring database schemas in the new cluster
                                                          ok
Adding ".old" suffix to old global/pg_control             ok

If you want to start the old cluster, you will need to remove
the ".old" suffix from
/var/lib/pgsql/15/data/global/pg_control.old.
Because "link" mode was used, the old cluster cannot be safely
started once the new cluster has been started.
Linking user relation files
                                                          ok
Setting next OID for new cluster                          ok
Sync data directory to disk                               ok
Creating script to delete old cluster                     ok
Checking for extension updates                            ok

Upgrade Complete
----------------
Optimizer statistics are not transferred by pg_upgrade.
Once you start the new server, consider running:
     /usr/pgsql-16/bin/vacuumdb --all --analyze-in-stages
Running this script will delete the old cluster's data files:
     ./delete_old_cluster.sh
```

You can expect this process to finish really quickly (usually within seconds).

VERIFY THE UPGRADE

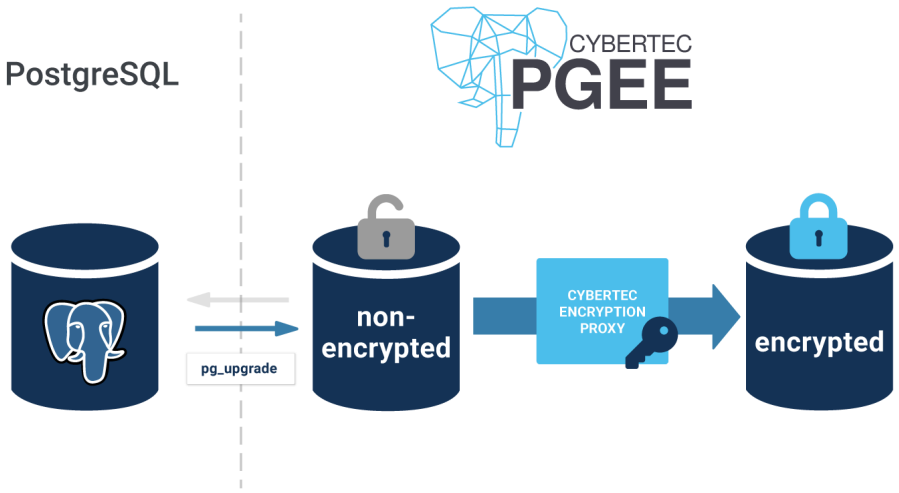
Let us verify the installation:

```
# sudo -u postgres psql
psql (16.4 EE 1.3.7)
Type "help" for help.

postgres=# \dt
           List of relations
 Schema |   Name    | Type  |  Owner
--------+-----------+-------+----------
 public | documents | table | postgres
(1 row)

postgres=# SELECT * FROM documents;
            doc
----------------------------
 My very important document
(1 row)
```

Voila, your PGEE deployment has been completed successfully.
At this point, the instance is running PGEE 16 and can be used by clients. Since we used `pg_upgrade`, which just copied the data files without changing them, the new instance is not encrypted yet. We use `repl_proxy` to encrypt the data in a second step.

STEP 3: ENCRYPTING YOUR PGEE INSTALLATION

Encrypting an existing database in PGEE is done by invoking a command called `repl_proxy`. It can easily be installed and will handle all replication and encryption / decryption related operations:



The following command installed the PGEE replication proxy:

```
# sudo yum install repl_proxy
```

Under the hood the replication proxy will attach to the PostgreSQL WAL and stream the WAL to the desired systems. It is therefore a good idea to create a user which is explicitly used for replication:

```
# sudo -u postgres psql
postgres=# CREATE USER replicator REPLICATION PASSWORD 'repl';
CREATE ROLE
```

Note that we want to use an existing database and encrypt it on the fly. Therefore we can quickly create a key (for demo purposes):

```
# KEY=$(dd if=/dev/random bs=1k count=1 | md5sum - \
      | cut -d ' ' -f 1)
1+0 records in
1+0 records out
1024 bytes (1.0 kB, 1.0 KiB) copied, 0.000334868 s, 3.1 MB/s
```

In real life we would of course hook up to a real KMS such as Keycloak, Kubernetes secrets or something along those lines.

STARTING THE REPLICATION PROXY

Once the replication proxy is installed and the key has been created we can start the proxy.

The syntax of the tool is as follows:

```
# repl_proxy --help
repl_proxy is a tool to modify data during replication.

Usage:
  repl_proxy [OPTION]...

Options:
  -h, --master-host=HOSTNAME    connect to master on this
                                host (default: "local socket")
  -p, --master-port=PORT             connect to master on this
                                port (default: "5432")
  -H, --proxy-host=HOSTNAME          run proxy on this host
                                (default: "localhost")
  -P, --proxy-port=PORT         run proxy on this port
                                (default "5433")
  -K, --encryption-key-command=COMMAND
                                command that returns
                                encryption key
  -k, --decryption-key-command=COMMAND
                                command that returns
                                decryption key
  -v, --verbose                 output additional debugging
                                information
  -?, --help                    show this help, then exit
```

Again, keep in mind: Normally you would use the CYBERTEC key management tool to secure key creation and use it as part of the -K command line option:

```
# sudo -u postgres repl_proxy -K "echo $KEY" &
repl_proxy: Starting socket on port 5433
```

In the next step we create a cluster to replicate our database into, encrypting everything on the way.

The trick here is: Normally pg_basebackup connects to the primary and streams the WAL. To encrypt an instance, however, we connect directly to the replication proxy and stream from there:

```
# sudo -u postgres pg_basebackup -h localhost -p 5433 \
      -U replicator -D /var/lib/pgsql/16/encr --verbose
Password: repl
pg_basebackup: initiating base backup, waiting for checkpoint
to complete
pg_basebackup: checkpoint completed
pg_basebackup: write-ahead log start point: 0/9000028 on
timeline 1
pg_basebackup: starting background WAL receiver
pg_basebackup: created temporary replication slot
"pg_basebackup_5836"
pg_basebackup: write-ahead log end point: 0/9000100
pg_basebackup: waiting for background process to finish
streaming ...
pg_basebackup: syncing data to disk ...
pg_basebackup: renaming backup_manifest.tmp to
backup_manifest
pg_basebackup: base backup completed
```

Finally we record the method to handle the encryption key (`encryption_key_command`) in postgresql.conf:

```
# echo "encryption_key_command = 'echo $KEY'" >>
/var/lib/pgsql/16/encr/postgresql.conf
```

After this process the replication proxy is not needed anymore as it is has its job. We can bring it back to the foreground and stop it:
Kill repl_proxy:

```
# fg
sudo -u postgres /usr/pgsql-16/bin/repl_proxy -K "echo $KEY"
^C
repl_proxy: Stopping server.
```

All there is left to do is to stop the old and start our freshly encrypted instance:

```
# sudo -u postgres /usr/pgsql-16/bin/pg_ctl \
     -D /var/lib/pgsql/16/data stop
# sudo -u postgres /usr/pgsql-16/bin/pg_ctl \
     -D /var/lib/pgsql/16/encr start
```

Let us verify our instance.

STEP 4: VERIFYING YOUR INSTALLATION

The encrypted PGEE instance has been started on port 5434. We can already connect to this port and verify the content of the database:

```
# sudo -u postgres psql
psql (16.4 EE 1.3.7)
Type "help" for help.

postgres=# \dt
           List of relations
 Schema |   Name    | Type  |  Owner
--------+-----------+-------+----------
 public | documents | table | postgres
(1 row)

postgres=# SELECT * FROM documents;
            doc
----------------------------
 My very important document
(1 row)

postgres=# SHOW data_encryption;
 data_encryption
-----------------
 on
(1 row)
```

As you can see the data is there and PostgreSQL shows that encryption has been enabled.

SUPPORT AND GETTING HELP

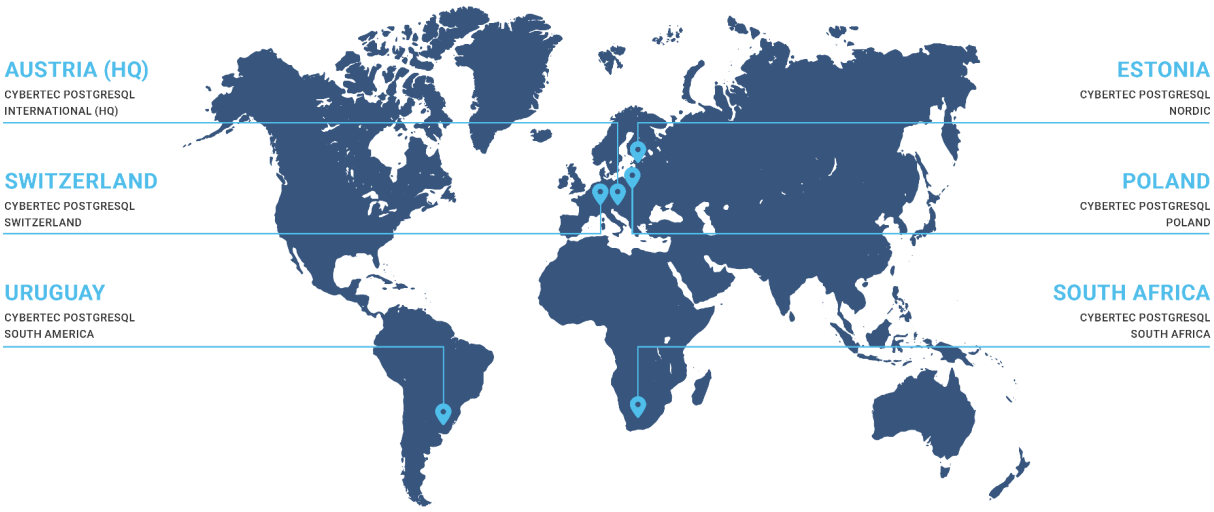
REQUESTING HELP

Thank you for using CYBERTEC PGEE and **thank you for being our customer.**
Your feedback is important to us and we are looking forward to hearing from you.
If you are facing any issues or technical questions please reach out to our technical team and make use of our 24x7 support and ticketing system.

 CYBERTEC Support Portal 

Our consultants are eager to help you with any technical and business related issues.

CYBERTEC
POSTGRESQL SERVICES & SUPPORT



CYBERTEC PostgreSQL International (HQ)
Gröhrmühlgasse 26
2700 Wiener Neustadt
Austria
Phone: +43 (0)2622 93022-0
office@cybertec.at

CYBERTEC PostgreSQL Nordic
Fahle Office
Tartu mnt 84a-M302
10112 Tallinn
Estonia
Phone: +372 712 3013
nordic@cybertec-postgresql.com

CYBERTEC PostgreSQL South America
Misiones 1486
oficina 301
11000 Montevideo
Uruguay
latam@cybertec-postgresql.com

CYBERTEC PostgreSQL Switzerland
Bahnhofstraße 10
8001 Zürich
Switzerland
Phone: +41 43 456 2684
swiss@cybertec-postgresql.com

CYBERTEC PostgreSQL Poland
Aleje Jerozolimskie 93
HubHub Nowogrodzka Square, 2nd floor
02-001 Warsaw
Poland
poland@cybertec-postgresql.com

CYBERTEC PostgreSQL South Africa
No. 26, Cambridge Office Park
5 Bauhinia Street, Highveld Techno Park
0046 Centurion
South Africa
Phone: +27(0)012 881 1911
africa@cybertec-postgresql.com

If you need further information

For more information, or if you have any questions about our range of products, tools and services, contact us. There's no obligation—send us an inquiry via email or give us a call.

ISO 27001 CERTIFIED

Contact

CYBERTEC PostgreSQL International GmbH
Römerstraße 19
2752 Wöllersdorf
AUSTRIA

+ 43 (0) 2622 93022-0
sales@cybertec-postgresql.com

VERSION HISTORY

Version	Effective Date	Description	Author	Reviewed By	Approved By
1.0	2024-11-09	Content written	Hans-Jürgen Schönig	Raj Verma	Christoph Berg