CYBERTEC
POSTGRESQL SERVICES & SUPPORT

CYBERTEC PGEE

Installation Guide for Debian / Ubuntu

Date: 2024-12-12
Publisher: CYBERTEC PGEE team

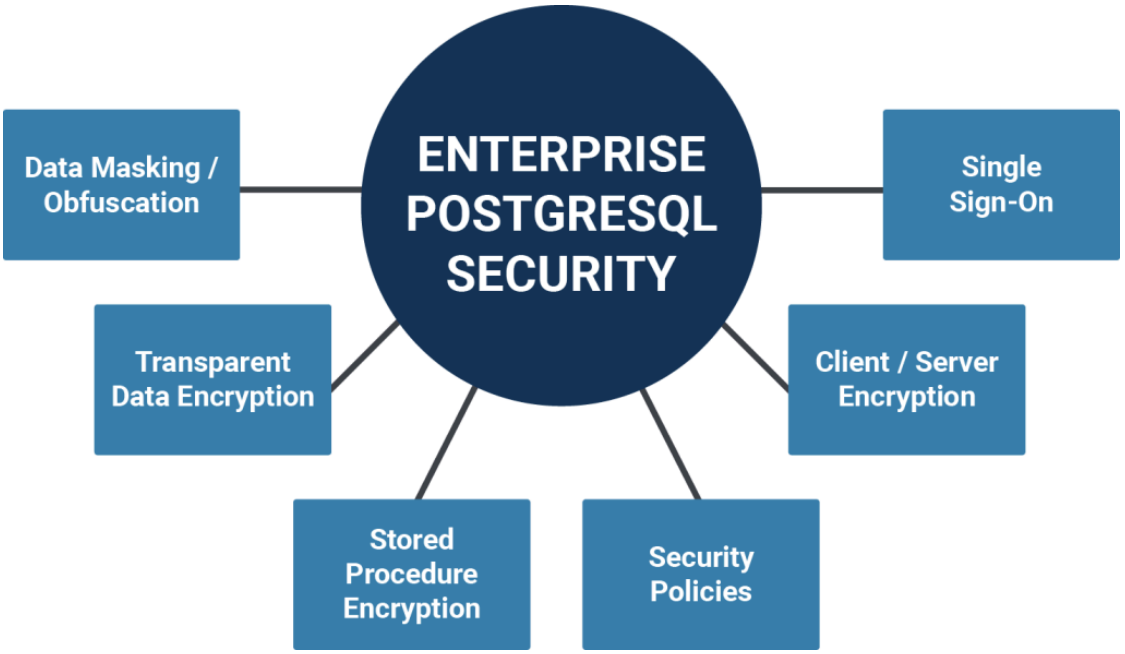
TABLE OF CONTENTS

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 **Debian / Ubuntu** based operating systems. The following operating systems are currently available and supported.

PGEE 16: SUPPORTED OPERATING SYSTEMS

- Debian 11 (bullseye) and 12 (bookworm) for x86_64
- Ubuntu 22.04 (jammy) and 24.04 (noble) for x86_64

* Additional operating systems and CPU-architectures are supported on request.

PGEE 17: SUPPORTED OPERATING SYSTEMS

- Debian 11 (bullseye) and 12 (bookworm) for x86_64
- Ubuntu 22.04 (jammy) and 24.04 (noble) for x86_64

* Additional operating systems and CPU-architectures are supported on request.

FAST PATH TO SUCCESS: PGEE INSTALLATION VIDEO

If you prefer to save some time, we have **prepared a tutorial video** for you.

It will explain how to:

- Install PGEE
- Encrypt an instance
- Verify for data

Watch the PGEE installation video

If you need more information this document will contain more details.

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 DEB REPOSITORY

The Debian / Ubuntu repositories can be found here:

https://deb.cybertec-postgresql.com/

Additional instructions can be found in the repository.

STEP 2: MAKE SURE CURL AND CA-CERTIFICATES ARE INSTALLED

Installing certificates can be achieved with the following command:

```
# sudo apt install curl ca-certificates
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
curl is already the newest version (7.88.1-10+deb12u7).
curl set to manually installed.
ca-certificates is already the newest version (20230311).
ca-certificates set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 0 not
upgraded.
```

In case those packages are not on your system they will be installed — otherwise Debian / Ubuntu will just inform us that nothing has to be done.

STEP 3: GET THE REPOSITORY KEY

The next steps ensure that the keys for the repository are installed. The next two commands facilitate this step:

```
# sudo install -d /etc/apt/keyrings/
# sudo curl -o /etc/apt/keyrings/cybertec-deb.asc --fail \
  https://deb.cybertec-postgresql.com/assets/cybertec-deb.asc
  % Total    % Received % Xferd  Average Speed   Time    Time
Time  Current
                                 Dload  Upload   Total
Spent    Left  Speed
100  3175  100  3175    0     0  30722      0 --:--:--
--:--:-- --:--:-- 31435
```

After this step we can move on to actually deploying the product.

STEP 4: INSTALL THE PUBLIC PGEE DEMO VERSION REPOSITORY

Once the keys have been deployed we can install the repository containing the PGEE packages:

```
# . /etc/os-release
# echo "deb [signed-by=/etc/apt/keyrings/cybertec-deb.asc] \
   https://deb.cybertec-postgresql.com/public
$VERSION_CODENAME main" \
   | sudo tee /etc/apt/sources.list.d/cybertec.list
deb [signed-by=/etc/apt/keyrings/cybertec-deb.asc]
https://deb.cybertec-postgresql.com/public bookworm main
```

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

STEP 5: INSTALL PACKAGES

First we have to update the list of packages in Debian, which can be achieved easily:

```
# sudo apt update
Hit:1 http://deb.debian.org/debian bookworm InRelease
Hit:2 http://deb.debian.org/debian bookworm-updates InRelease
Hit:3 http://security.debian.org/debian-security bookworm-security InRelease
Get:4 https://deb.cybertec-postgresql.com/public bookworm InRelease [4,369 B]
Get:5 https://deb.cybertec-postgresql.com/public bookworm/main amd64 Packages
[180 kB]
Fetched 184 kB in 1s (328 kB/s)
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
1 package can be upgraded. Run 'apt list --upgradable' to see it.
```

Finally the great moment has arrived: we can install our software!

HANDLING A MINIMAL INSTALLATION

The following command will deploy a minimal version of PGEE on your server:

```
# sudo apt install postgresql-16ee
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  libcommon-sense-perl libjson-perl libjson-xs-perl libllvm16
libpq5 libtypes-serialiser-perl libxslt1.1
postgresql-client-16ee postgresql-client-common
  postgresql-common postgresql-common-ee
Suggested packages:
  postgresql-doc-16ee
The following NEW packages will be installed:
  libcommon-sense-perl libjson-perl libjson-xs-perl libllvm16
libtypes-serialiser-perl libxslt1.1 postgresql-16ee
postgresql-client-16ee
  postgresql-client-common postgresql-common
postgresql-common-ee
The following packages will be upgraded:
  libpq5
1 upgraded, 11 newly installed, 0 to remove and 0 not
upgraded.
Need to get 43.7 MB of archives.
```

```
After this operation, 195 MB of additional disk space will be
used.
Do you want to continue? [Y/n]
Get:1 http://deb.debian.org/debian bookworm/main amd64
libjson-perl all 4.10000-1 [87.5 kB]
Get:2 http://deb.debian.org/debian bookworm/main amd64
libcommon-sense-perl amd64 3.75-3 [23.0 kB]

…

Fetched 43.7 MB in 1s (53.5 MB/s)
Reading changelogs... Done
Preconfiguring packages ...
Selecting previously unselected package libjson-perl.
(Reading database ... 107339 files and directories currently
installed.)

…

Setting up postgresql-common (262.pgee12~demo+1) ...

Creating config file
/etc/postgresql-common/createcluster.conf with new version
Building PostgreSQL dictionaries from installed
myspell/hunspell packages...
  en_us
  hu_hu
Removing obsolete dictionary files:
Created symlink
/etc/systemd/system/multi-user.target.wants/postgresql.servic
e → /lib/systemd/system/postgresql.service.
Setting up postgresql-common-ee (202408.1.pgee12~demo+1) ...
Setting up postgresql-16ee (16.4ee1.3.7-1.pgee12~demo+1) ...
Creating new PostgreSQL cluster 16/main ...
/usr/lib/postgresql/16/bin/initdb -D
/var/lib/postgresql/16/main --auth-local peer --auth-host
scram-sha-256 --no-instructions
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".

Data page checksums are disabled.
Data encryption is disabled.

fixing permissions on existing directory
/var/lib/postgresql/16/main ... ok
creating subdirectories ... ok
selecting dynamic shared memory implementation ... posix
selecting default max_connections ... 100
selecting default shared_buffers ... 128MB
selecting default time zone ... Europe/Vienna
creating configuration files ... ok
running bootstrap script ... ok
performing post-bootstrap initialization ... ok
syncing data to disk ... ok
update-alternatives: using
/usr/share/postgresql/16/man/man1/psql.1.gz to provide
/usr/share/man/man1/psql.1.gz (psql.1.gz) in auto mode
Processing triggers for man-db (2.11.2-2) ...
Processing triggers for libc-bin (2.36-9+deb12u8) ...
```

This example has shown how to install the most basic PGEE package. However, PGEE comes with a wide array of extensions, additional modules and packages which greatly increase your efficiency.

DEPLOYING A FULL VERSION OF PGEE

If you want to install the entire distribution consider the following command:

```
# sudo apt install postgresql-16ee-full
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
…
postgresql-16-hypopg postgresql-16-lockout
  postgresql-16-oracle-fdw postgresql-16-orafce
postgresql-16-pg-failover-slots postgresql-16-pg-permissions
postgresql-16-pg-qualstats
  postgresql-16-pg-stat-kcache
postgresql-16-pg-track-settings
postgresql-16-pg-wait-sampling postgresql-16-pgaudit
postgresql-16-pgextwlist
```

```
  postgresql-16-pgnodemx postgresql-16-pgvector
postgresql-16-pldebugger postgresql-16-plpgsql-sec
postgresql-16-postgis-3 postgresql-16-postgis-3-scripts
  postgresql-16-prohibit-commands postgresql-16-show-plans
postgresql-16-squeeze proj-data unixodbc-common
Suggested packages:
  geotiff-bin gdal-bin libgeotiff-epsg libhdf4-doc
libhdf4-alt-dev hdf4-tools liblcms2-utils odbc-postgresql
tdsodbc ogdi-bin proj-bin pgadmin3 | pgadmin4
  postgis
…
```

A variety of packages including Oracle compatibility, GIS modules (PostGIS), monitoring and a lot more will be installed. With the help of a simple command all dependencies will automatically be taken care of and magically installed on your platform.

STEP 6: CREATE AN ENCRYPTED CLUSTER

When installing PGEE we automatically create a non-encrypted, standard database installation (cluster). We do that to ensure that you can start immediately without having to worry about key management and other issues.

However, most people are looking for Transparent Data Encryption (TDE) and are therefore longing to enjoy the benefits of an encrypted database. Achieving full encryption can happen in a few simple steps.

First we have to stop the instance we have just created:

```
# sudo pg_dropcluster --stop 16 main
```

Key management in PGEE is flexible and allows for a lot of trickery. However, for the sake of simplicity we can create a random key and use it for the purpose of demonstration:

```
# 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
```

Let us use this key as follows:

NOTE: Do not use this method in production because the key will be displayed on the screen. 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 !

This is how we can start the instance. The -K option will accept the key by calling a program which returns the key to the infrastructure:

```
# sudo pg_createcluster --start 16 pgee -- -k -K "echo $KEY"
Creating new PostgreSQL cluster 16/pgee ...
/usr/lib/postgresql/16/bin/initdb -D
/var/lib/postgresql/16/pgee --auth-local peer --auth-host
scram-sha-256 --no-instructions -k -K echo
74ffca92369e08e52c6570620723653b
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".

Data page checksums are enabled.
Data encryption is enabled.

fixing permissions on existing directory
/var/lib/postgresql/16/pgee ... ok
creating subdirectories ... ok
selecting dynamic shared memory implementation ... posix
selecting default max_connections ... 100
selecting default shared_buffers ... 128MB
selecting default time zone ... Europe/Vienna
creating configuration files ... ok
running bootstrap script ... ok
performing post-bootstrap initialization ... ok
syncing data to disk ... ok
Ver Cluster Port Status Owner    Data directory
Log file
16  pgee    5432 online postgres /var/lib/postgresql/16/pgee
/var/log/postgresql/postgresql-16-pgee.log
```

After setting up the instance we can already move forward and connect to our freshly created CYBERTEC PGEE instance:

```
# sudo -u postgres psql
psql (16.4 EE 1.3.7 (Debian 16.4ee1.3.7-1.pgee12~demo+1),
server 16.4 EE 1.3.7 (Debian 16.4ee1.3.7-1.pgee12~demo+1))
```

```
Type "help" for help.
…
```

STEP 7: VERIFYING ENCRYPTION

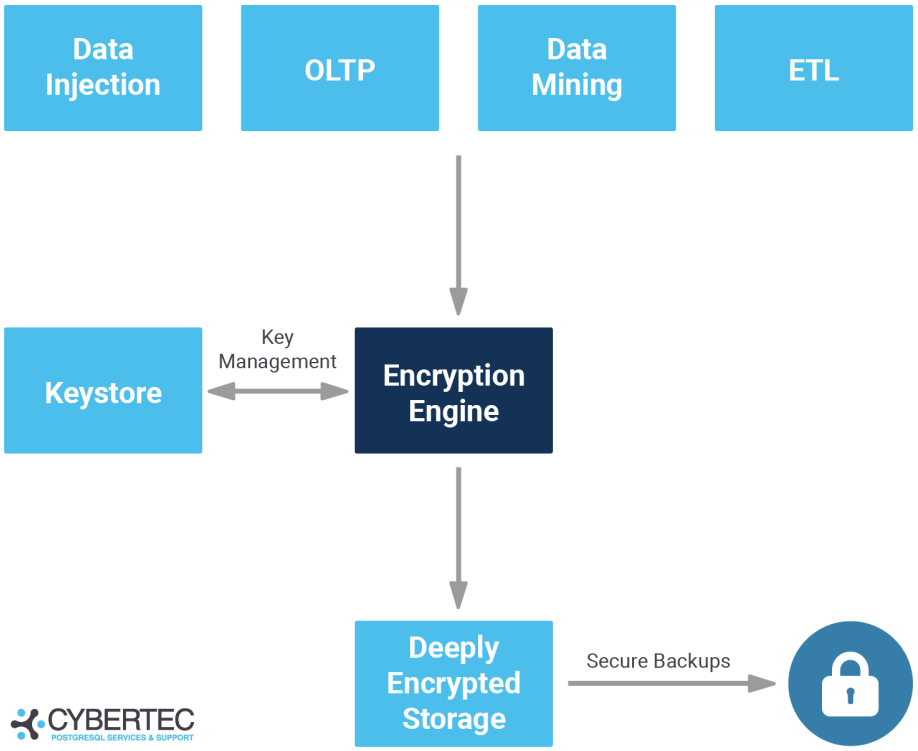
Of course we want to verify that encryption is working properly and as expected. By using the `SHOW` command we can easily check the status of our installation and ensure data is safe and secure:

```
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:

```
# apt list 2> /dev/null | grep postgresql-server
postgresql-server-dev-15/stable,stable-security 15.8-0+deb12u1 amd64
postgresql-server-dev-16ee/bookworm 16.4ee1.3.7-1.pgee12~demo+1 amd64
```

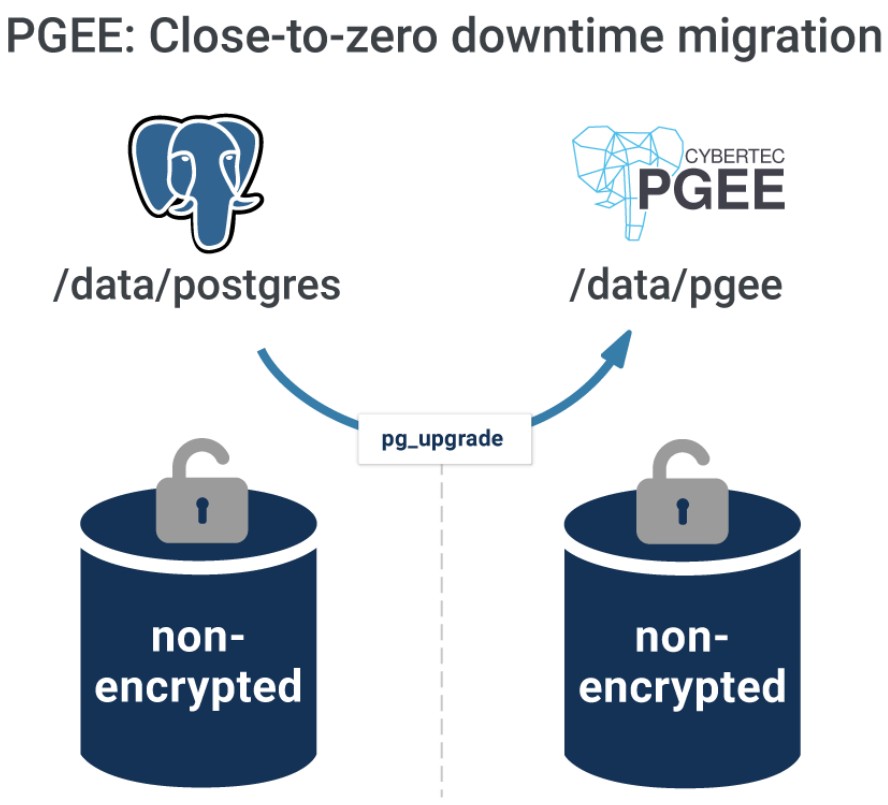
In this case we see PostgreSQL 15 (standard edition) as well as PGEE 16. Once this has been verified we can move on to the next step and start the migration process.

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
```

On Debian we can use `pg_upgradecluster` 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

```
# sudo pg_upgradecluster -m upgrade 15 main
Stopping old cluster...
Creating new PostgreSQL cluster 16/main ...
/usr/lib/postgresql/16/bin/initdb -D
/var/lib/postgresql/16/main --auth-local peer --auth-host
scram-sha-256 --no-instructions --encoding UTF8
     --lc-collate en_U
S.UTF-8 --lc-ctype en_US.UTF-8 --locale-provider libc
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 text search configuration will be set to
"english".

Data page checksums are disabled.
Data encryption is disabled.

fixing permissions on existing directory
/var/lib/postgresql/16/main ... ok
creating subdirectories ... ok
selecting dynamic shared memory implementation ... posix

…

Success. Please check that the upgraded cluster works. If it
does,
you can remove the old cluster with
    pg_dropcluster 15 main

Ver Cluster Port Status Owner    Data directory
15  main    5433 down   postgres /var/lib/postgresql/15/main …
16  main    5432 online postgres /var/lib/postgresql/16/main …
```

The `pg_upgradecluster` executable is part of the Debian / Ubuntu automation machinery and ensures that everything is wired up correctly under the hood. It will create the new instance and run all relevant scripts under the hood.

What is important to note here is that we are using the standard Debian process for upgrades. PGEE does not require any other steps.

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 options which creates hard links for the data files used by PGEE. In general this is the same process one would use on Debian for normal upgrades.

Here is how it works:

```
# sudo pg_upgradecluster -m link 15 main
```

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

VERIFY THE UPGRADE

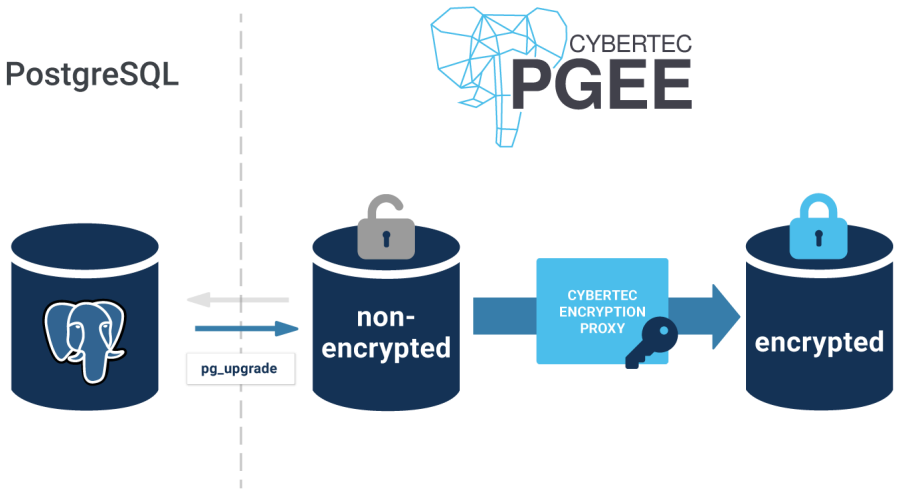
Let us verify the installation:

```
# sudo -u postgres psql
psql (16.4 EE 1.3.7 (Debian 16.4ee1.3.7-1.pgee12~demo+1),
server 16.4 EE 1.3.7 (Debian 16.4ee1.3.7-1.pgee12~demo+1))
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` (under the hood of `pg_upgradecluster`), 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 apt install postgresql-16-repl-proxy
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following NEW packages will be installed:
  postgresql-16-repl-proxy
0 upgraded, 1 newly installed, 0 to remove and 0 not
upgraded.
Need to get 52.2 kB of archives.
After this operation, 122 kB of additional disk space will be
used.
Get:1 https://deb.cybertec-postgresql.com/public
bookworm/main amd64 postgresql-16-repl-proxy amd64
1.0-1.pgee12~demo+1 [52.2 kB]
Fetched 52.2 kB in 1s (98.0 kB/s)
Selecting previously unselected package
postgresql-16-repl-proxy.
(Reading database ... 111641 files and directories currently
installed.)
Preparing to unpack
.../postgresql-16-repl-proxy_1.0-1.pgee12~demo+1_amd64.deb
...
```

```
Unpacking postgresql-16-repl-proxy (1.0-1.pgee12~demo+1) ...
Setting up postgresql-16-repl-proxy (1.0-1.pgee12~demo+1) ...
```

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 us as follows:

```
# /usr/lib/postgresql/16/bin/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 /usr/lib/postgresql/16/bin/repl_proxy \
     -K "echo $KEY" &
repl_proxy: Starting socket on port 5433
```

In the next step, we create an instance for Debiab to wire up all the config files. However, we do this only for Debian automation — we will throw it away immediately after creation because we want to rebuild the data directory from scratch.

```
# pg_createcluster 16 encr
Creating new PostgreSQL cluster 16/encr ...
/usr/lib/postgresql/16/bin/initdb -D
/var/lib/postgresql/16/encr --auth-local peer --auth-host \
      scram-sha-256 --no-instructions

…

Ver Cluster Port Status Owner    Data directory …
16  encr    5434 down   postgres /var/lib/postgresql/16/encr …
```

Everything has been wired up for Debian / Ubuntu so the next step is to replace the standard directory with an encrypted one:

```
# rm -rf /var/lib/postgresql/16/encr
```

The solution to this is pg_basebackup. 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/postgresql/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'" >>
/etc/postgresql/16/encr/postgresql.conf
```

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

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

All there is left to do is to start our freshly encrypted instance:

```
# pg_ctlcluster 16 encr start
```

The new instance will already show up in our list of instances:

```
# pg_lsclusters
Ver Cluster Port Status Owner    Data directory
15  main    5433 down   postgres /var/lib/postgresql/15/main
…
16  encr    5434 online postgres /var/lib/postgresql/16/encr
…
16  main    5432 online postgres /var/lib/postgresql/16/main
…
```

The list looks already promising but 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 -p 5434
psql (16.4 EE 1.3.7 (Debian 16.4ee1.3.7-1.pgee12~demo+1),
server 16.4 EE 1.3.7 (Debian 16.4ee1.3.7-1.pgee12~demo+1))
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.

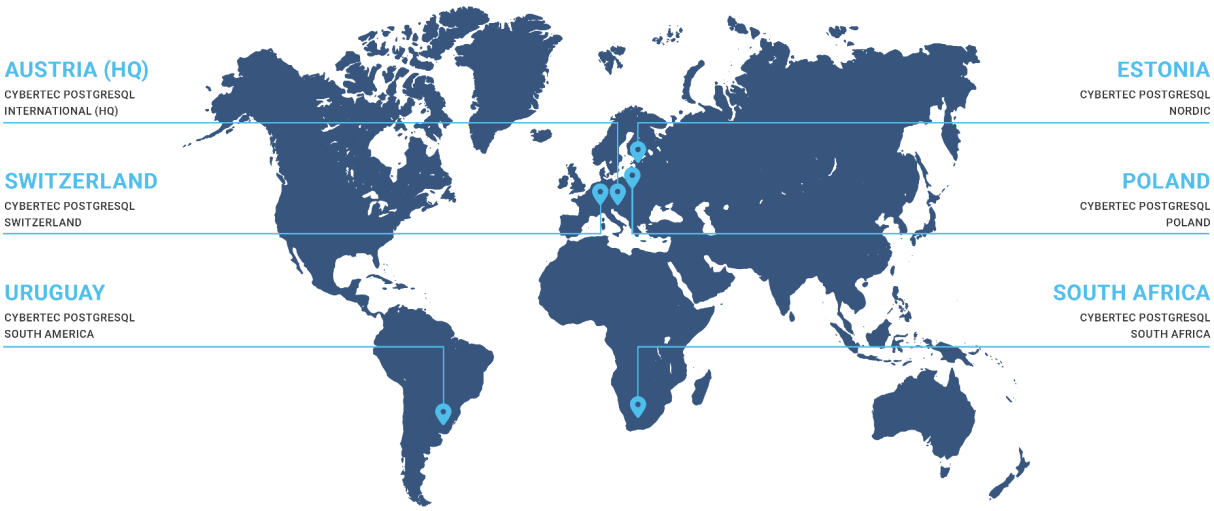
SUPPORT AND GETTING HELP

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 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	2024-12-12	Creating the document and finalizing the design	Hans-Jürgen Schönig	Christoph Berg	Maryia Bouraima