INSTALLING POSTGRESQL 12.X TDE FROM SOURCE

PostgreSQL 12.x TDE is a version of PostgreSQL which supports transparent data encryption. In many practical business cases it is necessary to encrypt data on disk. PostgreSQL TDE has been designed to do exactly that in the most efficient way possible.

This document describes how to install PostgreSQL TDE from source on Linux and Mac OS X.

1. Downloading the source code
2. Extracting and configuring PostgreSQL 12 TDE
3. Compiling the code
4. Setting up key management
5. Creating a database instance / cluster

1. DOWNLOADING THE SOURCE CODE

To download PostgreSQL TDE, you can go to our website[1] and download the latest tar file. If you prefer to use command line, you can also simply use "wget" to get the file. Here is an example:

```
wget https://download.cybertec-postgresql.com/postgresql-12.X-TDE-1.0beta2.tar.gz
```

2. EXTRACTING AND CONFIGURING POSTGRESQL 12 TDE

Once you have downloaded the file, you can easily unpack it. Here is how it works:

```
tar xvfz postgresql-12.X-TDE-1.0beta2.tar.gz
```

A directory will be created containing the entire source code of PostgreSQL TDE. Enter the directory and execute the following command:

For Mac OS X:

```
./configure --prefix=/Users/hs/pg12tde \
        --with-openssl \
        --with-perl \
        --with-python --with-ldap \
        CPPFLAGS="-I$(brew --prefix openssl)/include" \
        LDFLAGS="-L$(brew --prefix openssl)/lib"
```

[1] www.cybertec-postgresql.com/en/products/postgresql-transparent-data-encryption

> **Note:** On OS X, it makes sense to install HomeBrew to make sure that you can install all the dependencies first to compile the code. It is the easiest way to get all the software to compile and run PostgreSQL.

For Linux:
On Linux, compiling PostgreSQL is pretty straight forward. Make sure that all the dependencies outlined in the documentation[2] are met.
Then run:

```
./configure --prefix=/usr/local/pg12tde --with-openssl --with-perl \
      --with-python –with-ldap
```

If something goes wrong during configuring, it is very likely that you have missed a package.

3. COMPILING THE CODE

Once the "configure" step has been executed successfully, you can easily compile the code:

```
make install
cd contrib
make install
```

If you want to use more than one CPU core, you can add the -j flag to "make" to compile code in parallel.

Usually the code is compiled within 1 or 2 minutes.

PostgreSQL is now ready and you can already prepare your server infrastructure by adjusting the $PATH environment variables so you can easily access the database.

4. SETTING UP KEY MANAGEMENT

Key management is an important aspect. To encrypt a database instance, a key to do so has to come from somewhere. In case of PostgreSQL TDE, the key is coming from an external program which is totally flexible. Ideally the key DOES NOT COME from the local filesystem but from remote secure keystore.

Before creating your database instance, you have to write some code to make sure that the key can be read by the database during startup and instance creation.

[2] www.postgresql.org/docs/current/install-requirements.html

Here is the most simplistic example possible:

```
% cat /somewhere/provide_key.sh
#!/bin/sh

echo 882fb7c12e80280fd664c69d2d636913
```

All you need is a program that prints the key to stdout – and that's it! Make sure that PostgreSQL is able to execute this program:

```
% chmod +x /somewhere/provide_key.sh
```

> **Note:** You don't have to write a shell script – you can use any kind of executable such as a C, Go or Python.

5. CREATING A DATABASE INSTANCE / CLUSTER

Once the desired key management is in place, we can start to create the database instance. The beauty is that all it takes is a single line and PostgreSQL will do all the magic for you:

```
% initdb -D /some_path/db12tde -K /somewhere/provide_key.sh
The files belonging to this database system will be owned by user "hs".
This user must also own the server process.

The database cluster will be initialized with locale "C".
The default database encoding has accordingly been set to "SQL_ASCII".
The default text search configuration will be set to "english".

Data page checksums are disabled.
Data encryption is enabled.

creating directory /some_path/db12tde ... 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/Berlin
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
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 /some_path/db12tde -l logfile start
```

The difference between PostgreSQL and PostgreSQL TDE is that there is an optional -K option. Otherwise there is no difference. If you pass the name of your key management executable to initdb, all the magic will happen automatically. There is nothing more to do and you can normally start the database:

```
% pg_ctl -D /some_path/db12tde start
2020-01-29 11:54:19.131 CET [42193] LOG:  starting PostgreSQL 12.X-TDE-1.0beta2 on
x86_64-apple-darwin19.2.0, compiled by Apple clang version 11.0.0 (clang-1100.0.33.17),
64-bit
2020-01-29 11:54:19.132 CET [42193] LOG:  listening on IPv6 address "::1", port 5432
2020-01-29 11:54:19.132 CET [42193] LOG:  listening on IPv4 address "127.0.0.1", port
5432
2020-01-29 11:54:19.133 CET [42193] LOG:  listening on Unix socket "/tmp/.s.PGSQL.5432"
waiting for server to start.…
2020-01-29 11:54:19.151 CET [42197] LOG:  database system was shut down at 2020-01-29
11:54:05 CET
2020-01-29 11:54:19.154 CET [42193] LOG:  database system is ready to accept connections
 done
server started
```

There is no need for additional parameters. Everything has been wired for you in the background already.

ENCRYPTION BEHIND THE SCENES

The reason why you don't need additional parameters is that initdb has already added the configuration to postgresql.conf for you:

```
% grep encryption_key postgresql.conf
encryption_key_command = '/somewhere/provide_key.sh'
```

postgresql.conf already has the information to fetch the key on startup. The advantage is that you don't have to adapt existing scripts. The only change you have to make is to add one more parameter to initdb. Et voilá – you are done.

IS MY DATABASE SERVER ENCRYPTED?

The most obvious and most common question at this point is: Is that really everything? How can I figure out that my server is really encrypted? To do that, you can make use of pg_controldata which is a program extracting some useful information out of your database server. Keep in mind that pg_controldata has NOTHING to do with TDE – it is there anyway.

In case of PostgreSQL TDE, pg_controldata produces slightly more information than usual:

```
% pg_controldata /some_where/ | grep -i encryp
Data encryption:           on
Data encryption fingerprint:          740A905130FE614CE0BE36B612157A09
```

As you can see, encryption is on and your data is secure.
However, if you don't have access to the machine via SSH, there is a second way to see if data is encrypted:

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

PostgreSQL TDE offers an additional variable. data_encryption on / off will also tell you whether the database has been encrypted or not.

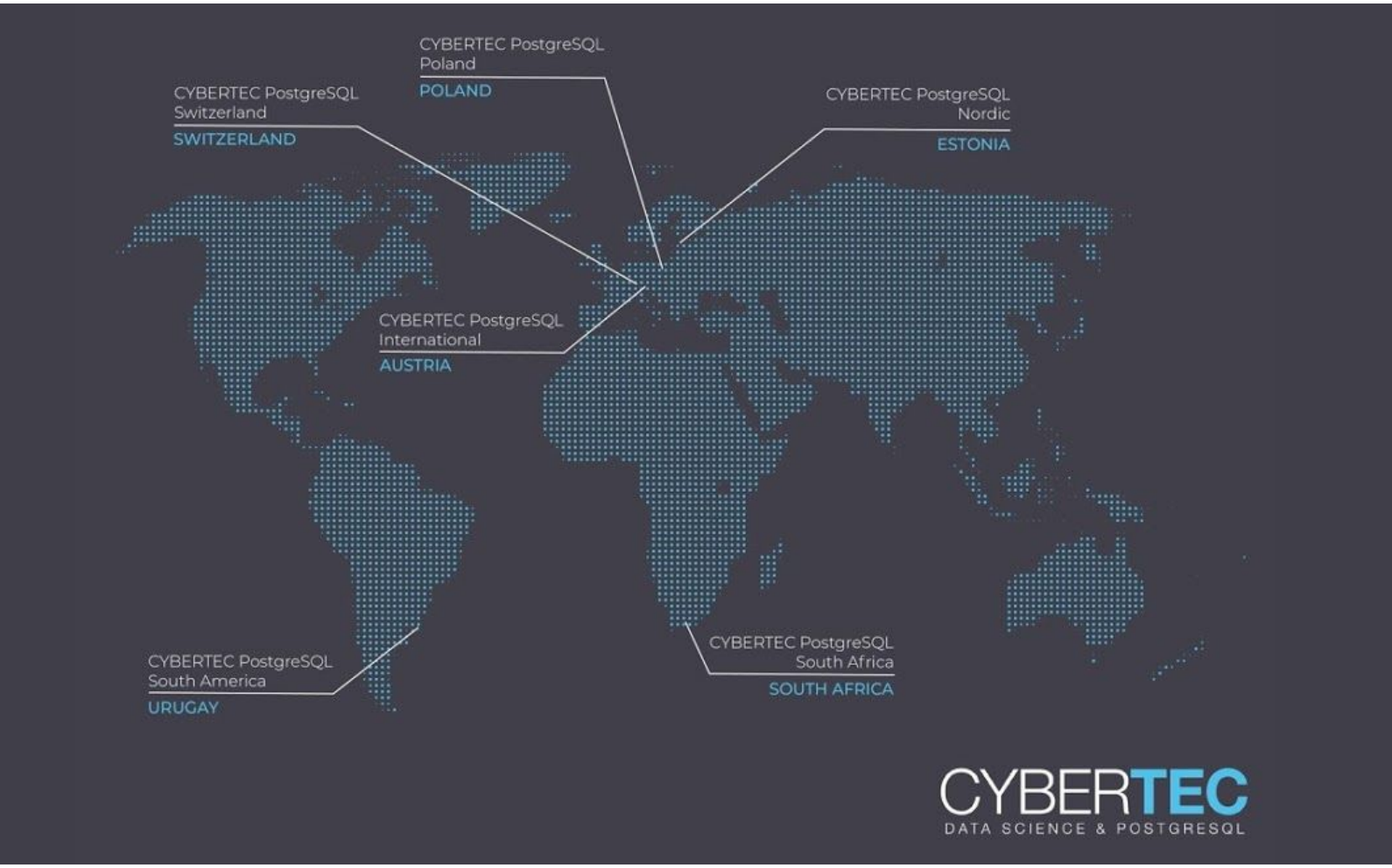
MORE INFORMATION

If you want to learn more, we recommend visiting our website on a regular basis and follow us on Twitter (@PostgresSupport). We will keep you updated on recent developments. Also keep in mind that CYBERTEC provides commercial 24x7 support for PostgreSQL TDE. **Please report bugs to bugs-tde@cybertec.at!**

LET'S KEEP IN TOUCH!

www.cybertec-postgresql.com
twitter.com/PostgresSupport
github.com/cybertec-postgresql
www.facebook.com/cybertec.postgresql
www.linkedin.com/company/cybertec-sch-nig-&-sch-nig-gmbh
office@cybertec.at

CONTACT



CYBERTEC PostgreSQL International (HQ)
Gröhrmühlgasse 26
2700 Wiener Neustadt
Austria

+43 (0)2622 93022-0
office@cybertec.at

CYBERTEC PostgreSQL Switzerland
Bahnhofstraße 10
8001 Zürich
Switzerland

+41 43 456 2684
office@cybertec.at

CYBERTEC PostgreSQL South America
Misiones 1486 oficina 301
11000 Montevideo
Uruguay

+43 (0)2622 93022-0
office@cybertec.at

CYBERTEC PostgreSQL Nordic
Fahle Office
Tartu mnt 84a-M302
10112 Tallinn
Estonia

+372 53070910
office@cybertec.at

CYBERTEC PostgreSQL Poland
Aleje Jerozolimskie 93
HubHub Nowogrodzka
Square, 2nd floor
02-001 Warsaw
Poland

+43 (0)2622 93022-0
office@cybertec.at

CYBERTEC PostgreSQL South Africa
No. 26, Cambridge Office Park
5 Bauhinia Street, Highveld
Techno Park
0046 Centurion
South Africa

+27(0)76 708 7484
africa@cybertec.at

CYBERTEC WORLDWIDE

AUSTRIA | SWITZERLAND | ESTONIA | POLAND | URUGUAY | SOUTH AFRICA