Module B

Installation de PostgreSQL

DALIBO
L'expertise PostgreSQL

22.09

Dalibo SCOP

https://dalibo.com/formations

Installation de PostgreSQL

Module B

TITRE : Installation de PostgreSQL
SOUS-TITRE : Module B

REVISION: 22.09
DATE: 02 septembre 2022

Remerciements : Ce manuel de formation est une aventure collective qui se transmet au sein de notre société depuis des années. Nous remercions chaleureusement ici toutes les personnes qui ont contribué directement ou indirectement à cet ouvrage, notamment : Jean-Paul Argudo, Alexandre Anriot, Carole Arnaud, Alexandre Baron, David Bidoc, Sharon Bonan, Franck Boudehen, Arnaud Bruniquel, Damien Clochard, Christophe Courtois, Marc Cousin, Gilles Darold, Jehan-Guillaume de Rorthais, Ronan Dunklau, Vik Fearing, Stefan Fercot, Pierre Giraud, Nicolas Gollet, Dimitri Fontaine, Florent Jardin, Virginie Jourdan, Luc Lamarle, Denis Laxalde, Guillaume Lelarge, Benoit Lobréau, Jean-Louis Louër, Thibaut Madelaine, Adrien Nayrat, Alexandre Pereira, Flavie Perette, Robin Portigliatti, Thomas Reiss, Maël Rimbault, Julien Rouhaud, Stéphane Schildknecht, Julien Tachoires, Nicolas Thauvin, Be Hai Tran, Christophe Truffier, Cédric Villemain, Thibaud Walkowiak, Frédéric Yhuel.

LICENCE CREATIVE COMMONS BY-NC-SA 2.0 FR

Attribution - Pas d'Utilisation Commerciale - Partage dans les Mêmes Conditions

Vous êtes autorisé à :

- Partager, copier, distribuer et communiquer le matériel par tous moyens et sous tous formats
- Adapter, remixer, transformer et créer à partir du matériel

Dalibo ne peut retirer les autorisations concédées par la licence tant que vous appliquez les termes de cette licence selon les conditions suivantes :

Attribution : Vous devez créditer l'œuvre, intégrer un lien vers la licence et indiquer si des modifications ont été effectuées à l'œuvre. Vous devez indiquer ces informations par tous les moyens raisonnables, sans toutefois suggérer que Dalibo vous soutient ou soutient la façon dont vous avez utilisé ce document.

Pas d'Utilisation Commerciale : Vous n'êtes pas autorisé à faire un usage commercial de ce document, tout ou partie du matériel le composant.

Partage dans les Mêmes Conditions : Dans le cas où vous effectuez un remix, que vous transformez, ou créez à partir du matériel composant le document original, vous devez diffuser le document modifié dans les même conditions, c'est à dire avec la même licence avec laquelle le document original a été diffusé.

Pas de restrictions complémentaires : Vous n'êtes pas autorisé à appliquer des conditions légales ou des mesures techniques qui restreindraient légalement autrui à utiliser le document dans les conditions décrites par la licence.

Note : Ceci est un résumé de la licence. Le texte complet est disponible ici :

https://creativecommons.org/licenses/by-nc-sa/2.0/fr/legalcode

Pour toute demande au sujet des conditions d'utilisation de ce document, envoyez vos questions à contact@dalibo.com[1] !

[1] mailto:contact@dalibo.com

Chers lectrices & lecteurs,

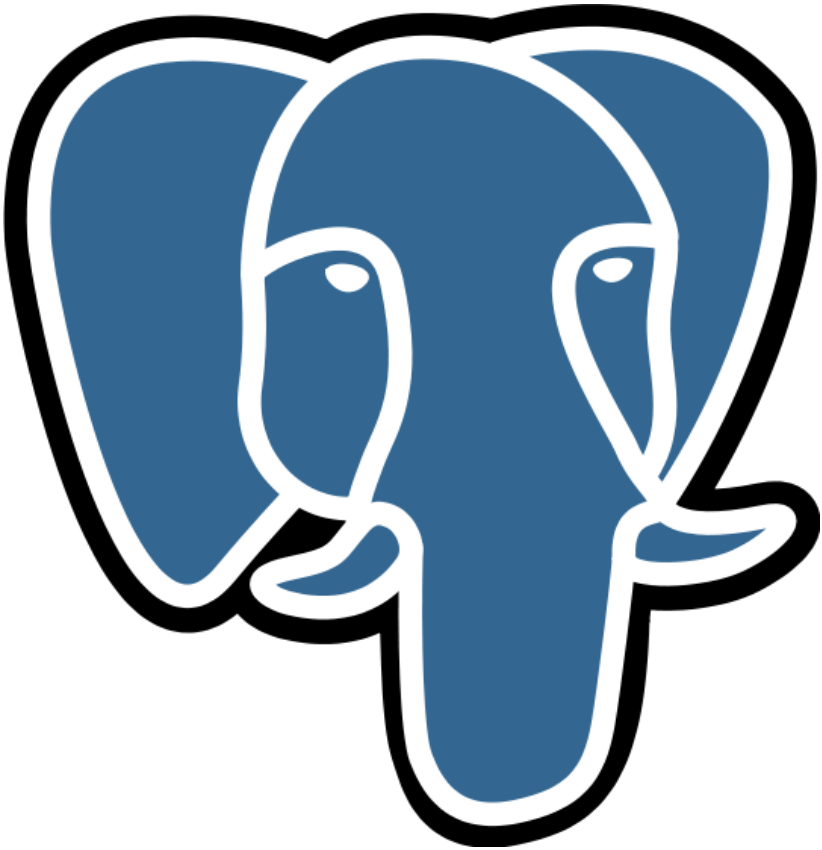
Nos formations PostgreSQL sont issues de nombreuses années d'études, d'expérience de terrain et de passion pour les logiciels libres. Pour Dalibo, l'utilisation de PostgreSQL n'est pas une marque d'opportunisme commercial, mais l'expression d'un engagement de longue date. Le choix de l'Open Source est aussi le choix de l'implication dans la communauté du logiciel.

Au-delà du contenu technique en lui-même, notre intention est de transmettre les valeurs qui animent et unissent les développeurs de PostgreSQL depuis toujours : partage, ouverture, transparence, créativité, dynamisme... Le but premier de nos formations est de vous aider à mieux exploiter toute la puissance de PostgreSQL mais nous espérons également qu'elles vous inciteront à devenir un membre actif de la communauté en partageant à votre tour le savoir-faire que vous aurez acquis avec nous.

Nous mettons un point d'honneur à maintenir nos manuels à jour, avec des informations précises et des exemples détaillés. Toutefois malgré nos efforts et nos multiples relectures, il est probable que ce document contienne des oublis, des coquilles, des imprécisions ou des erreurs. Si vous constatez un souci, n'hésitez pas à le signaler via l'adresse formation@dalibo.com !

Table des Matières

1 INSTALLATION DE POSTGRESQL



1.1 INTRODUCTION

- Installation depuis les sources
- Installation depuis les binaires
  - installation à partir des paquets
  - installation sous Windows
- Premiers réglages
- Mises à jours

Il existe trois façons d'installer PostgreSQL :

- Les installeurs graphiques :
  - avantages : installation facile, idéale pour les nouveaux venus ;
  - inconvénients : pas d'intégration avec le système de paquets du système d'exploitation.
- les paquets du système :
  - avantages : meilleure intégration avec les autres logiciels, idéal pour un serveur en production ;
  - inconvénients : aucun ?
- Le code source :
  - avantages : configuration très fine, ajout de patchs, intéressant pour les utilisateurs expérimentés et les testeurs ;
  - inconvénients : nécessite un environnement de compilation, ainsi que de configurer utilisateurs et script de démarrage.

Nous allons maintenant détailler chaque façon d'installer PostgreSQL.

1.2 INSTALLATION À PARTIR DES SOURCES

Étapes :
- Téléchargement
- Vérification des prérequis
- Compilation
- Installation

Nous allons aborder ici les différentes étapes à réaliser pour installer PostgreSQL à partir des sources :

- trouver les fichiers sources ;
- préparer le serveur pour accueillir PostgreSQL ;
- compiler le serveur ;
- vérifier le résultat de la compilation ;
- installer les fichiers compilés.

1.2.1 TÉLÉCHARGEMENT

- Disponible via http
- Télécharger le fichier `postgresql-<version>.tar.bz2`

Les fichiers sources et les instructions de compilation sont disponibles sur le site officiel du projet[2] . L'ancien serveur FTP anonyme a été remplacé par https://ftp.postgresql.org/pub/. Le nom du fichier à télécharger se présente toujours sous la forme `postgresql-<version>.tar.bz2` où `<version>` représente la version de PostgreSQL.

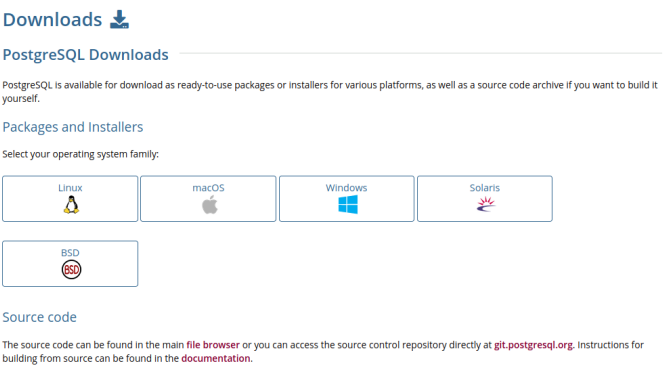
Lorsque la future version du logiciel est en phase de test (versions bêta), les sources sont accessibles à l'adresse suivante : https://www.postgresql.org/developer/beta.

Voici comment récupérer la dernière version des sources de PostgreSQL :

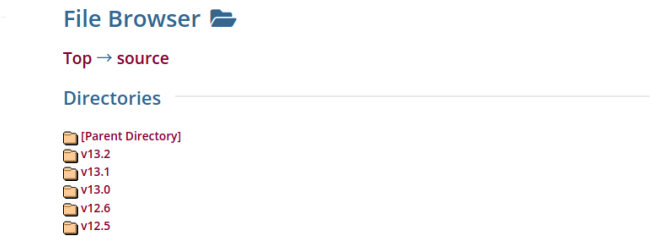
[2] https://www.postgresql.org/download/

- Se rendre sur la page d'accueil du projet PostgreSQL ;

- Cliquer sur le lien *Downloads* ;

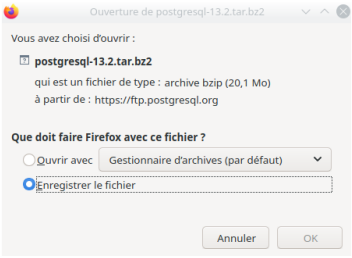


- Dans *Source code*, cliquer sur le lien *file browser* de la partie ;



- Cliquer sur le lien « v13.2 » (la 13.2 était la dernière version stable disponible lors de la mise à jour de cette présentation ; utilisez la version stable la plus récente possible) ;

- Cliquer sur le lien « postgresql-13.2.tar.bz2 » et enregistrer le fichier



1.2.2 PHASES DE COMPILATION/INSTALLATION

- Processus standard :

```
$ tar xvfj postgresql-<version>.tar.bz2
$ cd postgresql-<version>
$ ./configure
$ make
# make install
```

La compilation de PostgreSQL suit un processus classique.

Comme pour tout programme fourni sous forme d'archive tar, nous commençons par décompacter l'archive dans un répertoire. Le répertoire de destination pourra être celui de l'utilisateur **postgres** (~) ou bien dans un répertoire partagé dédié aux sources (`/usr/src/postgres` par exemple) afin de donner l'accès aux sources du programme ainsi qu'à la documentation à tous les utilisateurs du système.

```
$ cd ~
$ tar xvfj postgresql-<version>.tar.bz2
```

Une fois l'archive extraite, il faut dans un premier temps lancer le script d'auto-configuration des sources. **Attention aux options de configuration !**

```
$ cd postgresql-<version>
$ ./configure [OPTIONS]
```

Les dernières lignes de la phase de configuration doivent correspondre à la création d'un certain nombre de fichiers, dont notamment le Makefile :

```
configure: creating ./config.status
config.status: creating GNUmakefile
config.status: creating src/Makefile.global
config.status: creating src/include/pg_config.h
config.status: creating src/include/pg_config_ext.h
config.status: creating src/interfaces/ecpg/include/ecpg_config.h
config.status: linking src/backend/port/tas/dummy.s to src/backend/port/tas.s
config.status: linking src/backend/port/posix_sema.c to src/backend/port/pg_sema.c
config.status: linking src/backend/port/sysv_shmem.c to src/backend/port/pg_shmem.c
config.status: linking src/include/port/linux.h to src/include/pg_config_os.h
config.status: linking src/makefiles/Makefile.linux to src/Makefile.port
```

Vient ensuite la phase de compilation des sources de PostgreSQL pour en construire les différents exécutables :

```
$ make
```

Cette phase est la plus longue. Cependant, cela reste assez rapide sur du matériel récent. Il est possible de le rendre plus rapide en utilisant une compilation parallélisée grâce à

l'option `--jobs`.

Sur certains systèmes, comme Solaris, AIX ou les BSD, la commande `make` issue des outils GNU s'appelle en fait `gmake`. Sous Linux, elle est habituellement renommée en `make`.

Si une erreur s'est produite, il est nécessaire de la corriger avant de continuer. Sinon, il est possible d'installer le résultat de la compilation :

```
# make install
```

Cette commande installe les fichiers dans les répertoires spécifiés à l'étape de configuration, notamment via l'option `--prefix`. Sans précision dans l'étape de configuration, les fichiers sont installés dans le répertoire `/usr/local/pgsql`.

1.2.3 OPTIONS POUR ./CONFIGURE

- Quelques options de configuration notables :
  - `--prefix=`**répertoire**
  - `--with-pgport=`**port**
  - `--with-openssl`
  - `--enable-nls`
  - `--with-perl`
- Pour retrouver les options de compilation à postériori

```
$ pg_config --configure
```

Le script de configuration de la compilation possède un certain nombre de paramètres optionnels. Parmi toutes ces options, citons les suivantes qui sont probablement les plus intéressantes :

- `--prefix=`**répertoire** : permet de définir un répertoire d'installation personnalisé (par défaut, il s'agit de `/usr/local/pgsql`) ;
- `--with-pgport`=**port** : permet de définir un port par défaut différent de 5432 ;
- `--with-openssl` : permet d'activer le support d'OpenSSL pour bénéficier de connexions chiffrées ;
- `--enable-nls` : permet d'activer le support de la langue utilisateur pour les messages provenant du serveur et des applications ;
- `--with-perl` : permet d'installer le langage de routines PL/perl. PostgreSQL.

En cas de compilation pour la mise à jour d'une version déjà installée, il est important de connaître les options utilisées lors de la précédente compilation. La personne qui a procédé à cette compilation n'est pas forcément là, n'a pas forcément conservé cette

information, et il faut donc un autre moyen pour récupérer cette information. L'outil `pg_config` le permet ainsi :

```
$ pg_config --configure
  '--prefix=/usr'
  '--mandir=/usr/share/postgresql/12/man'
  '--with-docdir=/usr/share/doc/postgresql-doc-12'
  '--datadir=/usr/share/postgresql/12'
  '--bindir=/usr/lib/postgresql/12/bin'
  '--libdir=/usr/lib/postgresql/12/lib'
  '--includedir=/usr/include/postgresql/12'
  '--enable-nls'
  '--enable-integer-datetimes'
  '--enable-thread-safety'
  '--enable-debug' '--disable-rpath' '--with-tcl'
  '--with-perl' '--with-python' '--with-pam'
  '--with-krb5' '--with-openssl' '--with-gnu-ld'
  '--with-tclconfig=/usr/lib/tcl8.5'
  '--with-tkconfig=/usr/lib/tk8.5'
  '--with-includes=/usr/include/tcl8.5'
  '--with-system-tzdata=/usr/share/zoneinfo'
  '--sysconfdir=/etc/postgresql-common'
  '--with-gssapi' '--with-libxml'
  '--with-libxslt' '--with-ldap' '--with-ossp-uuid'
  'CFLAGS= -fPIC' 'LDFLAGS= -Wl,--as-needed'
```

1.2.4 TESTS DE NON RÉGRESSION

- Exécution de tests unitaires
- Permet de vérifier l'état des exécutables construits
- Action `check` de la commande `make`

```
$ make check
```

Il est possible d'effectuer des tests avec les exécutables fraîchement construits grâce à la commande suivante :

```
$ make check
[...]
rm -rf ./testtablespace
mkdir ./testtablespace
PATH="/home/dalibo/git.postgresql/tmp_install/usr/local/pgsql/bin:$PATH"
  LD_LIBRARY_PATH="/home/dalibo/git.postgresql/tmp_install/usr/local/pgsql/lib"
  ../../../src/test/regress/pg_regress --temp-instance=./tmp_check --inputdir=.
  --bindir=  --dlpath=. --max-concurrent-tests=20
  --schedule=./parallel_schedule
```

```
============== creating temporary instance            ==============
============== initializing database system           ==============
============== starting postmaster                    ==============
running on port 60849 with PID 31852
============== creating database "regression"         ==============
CREATE DATABASE
ALTER DATABASE
============== running regression test queries        ==============
test tablespace                   ... ok          134 ms
parallel group (20 tests):  char varchar text boolean float8 name money pg_lsn
       float4 oid uuid txid bit regproc int2 int8 int4 enum numeric rangetypes
     boolean                      ... ok           43 ms
     char                         ... ok           25 ms
     name                         ... ok           60 ms
     varchar                      ... ok           25 ms
     text                         ... ok           39 ms
[...]
     partition_join               ... ok          835 ms
     partition_prune              ... ok          793 ms
     reloptions                   ... ok           73 ms
     hash_part                    ... ok           43 ms
     indexing                     ... ok          828 ms
     partition_aggregate          ... ok          799 ms
     partition_info               ... ok          106 ms
     tuplesort                    ... ok         1137 ms
     explain                      ... ok           80 ms
test event_trigger                ... ok           64 ms
test fast_default                 ... ok           89 ms
test stats                        ... ok          571 ms
============== shutting down postmaster               ==============
============== removing temporary instance            ==============

======================
 All 201 tests passed.
======================
[...]
```

Les tests de non régression sont une suite de tests qui vérifient que PostgreSQL fonctionne correctement sur la machine cible. Ces tests ne peuvent pas être exécutés en tant qu'utilisateur **root**. Le fichier `src/test/regress/README` et la documentation contiennent des détails sur l'interprétation des résultats de ces tests.

1.2.5 CRÉATION DE L'UTILISATEUR

- Jamais **root**
- Ajout d'un utilisateur :
  - lancera PostgreSQL
  - sera le propriétaire des répertoires et fichiers
- Variables d'environnement :

```
export PATH=/usr/local/pgsql/bin:$PATH
export LD_LIBRARY_PATH=/usr/local/pgsql/lib:$LD_LIBRARY_PATH
export MANPATH=$MANPATH:/usr/local/pgsql/share/man
export PGDATA=/usr/local/pgsql/data
```

Le serveur PostgreSQL ne peut pas être exécuté par l'utilisateur **root**. Pour des raisons de sécurité, il est nécessaire de passer par un utilisateur sans droits particuliers. Cet utilisateur sera le seul propriétaire des répertoires et fichiers gérés par le serveur PostgreSQL. Il sera aussi le compte qui permettra de lancer PostgreSQL.

Cet utilisateur est généralement appelé **postgres** mais ce n'est pas une obligation. Une façon de distinguer différentes instances installées sur le même serveur physique ou virtuel est d'utiliser un compte différent par instance, surtout si l'on utilise les variables d'environnement. (Avec un seul compte système, il est facile de nommer les instances avec le paramètre `cluster_name`, dont le contenu apparaîtra dans les noms des processus.)

Il peut aussi se révéler nécessaire de positionner un certain nombre de variables d'environnement.

Afin de rendre l'exécution de PostgreSQL possible à partir de n'importe quel répertoire, il est très pratique (essentiel ?) d'ajouter le répertoire d'installation des exécutables (`/usr/local/pgsql/bin` par défaut ou le chemin indiqué à l'option `--prefix` lors de l'étape `configure`) aux chemins de recherche. Pour cela, nous modifions la variable d'environnement `PATH`. En complément, la variable `LD_LIBRARY_PATH` indique au système où trouver les différentes bibliothèques nécessaires à l'exécution des programmes. Voici un exemple avec ces deux variables :

```
export PATH=/usr/local/pgsql/bin:$PATH
export LD_LIBRARY_PATH=/usr/local/pgsql/lib:$LD_LIBRARY_PATH
```

D'autres variables d'environnement peuvent éventuellement être utiles:

- `MANPATH` pour ajouter aux chemins de recherche des pages de manuel celui de votre installation de PostgreSQL ;
- `PGDATA` qui sera consulté par certains exécutables de PostgreSQL pour déterminer où se trouve le répertoire de travail de l'instance.

Voici un exemple avec ces deux variables :

```
export MANPATH=$MANPATH:/usr/local/pgsql/share/man
export PGDATA=/usr/local/pgsql/data
```

Afin de vous éviter d'avoir à redéfinir ces variables d'environnement après chaque redémarrage du système, il est conseillé d'inclure ces lignes dans votre fichier `${HOME}/.profile` ou dans `/etc/profile`.

1.2.6 CRÉATION DU RÉPERTOIRE DE DONNÉES : INITDB

```
$ initdb -D /usr/local/pgsql/data
```

- Une seule instance !
- Options d'emplacement :
  - `--data` pour les fichiers de données
  - `--waldir` pour les journaux de transactions
- Options de sécurité :
  - `--pwprompt` : configurer immédiatement le mot de passe de l'utilisateur **postgres**
  - `--data-checksums` : sommes de contrôle (conseillé !)
  - `--allow-group-access` : droits 750 sur le répertoire
- Autre option :
  - `--wal-segsize` : taille personnalisée des journaux de transactions

La commande `initdb` doit être exécutée sous le compte de l'utilisateur système PostgreSQL décrit dans la section précédente (généralement **postgres**).

Elle permet de créer les fichiers d'une nouvelle instance avec une première base dans le répertoire indiqué.

Ce répertoire ne doit être utilisé que par une seule instance (processus) à la fois ! PostgreSQL vérifie au démarrage qu'aucune autre instance du même serveur n'utilise les fichiers indiqués, mais cette protection n'est pas absolue, notamment avec des accès depuis des systèmes différents.
Faites donc bien attention de ne lancer PostgreSQL qu'une seule fois sur un répertoire de données.

Si le répertoire n'existe pas, `initdb` tentera de le créer, s'il a les droits pour le faire. S'il existe, il doit être vide.

Attention : pour des raisons de sécurité et de fiabilité, les répertoires choisis pour les données de votre instance **ne doivent pas** être à la racine d'un point de montage. Que ce soit le répertoire PGDATA, le répertoire `pg_wal` ou les éventuels *tablespaces*. Si

un ou plusieurs points de montage sont dédiés à l'utilisation de PostgreSQL, positionnez toujours les données dans un sous-répertoire, voire deux niveaux en-dessous du point de montage, comme dans cette norme courante : `<point de montage>/<version majeure>/<nom instance>`.
À ce propos, voir :

- chapitre *Use of Secondary File Systems*[a] ;
- le détail des raisons techniques[b] .

Voici ce qu'affiche cette commande :

```
$ initdb --data /usr/local/pgsql/data --data-checksums

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 locales
  COLLATE:  en_US.UTF-8
  CTYPE:    en_US.UTF-8
  MESSAGES: en_US.UTF-8
  MONETARY: fr_FR.UTF-8
  NUMERIC:  fr_FR.UTF-8
  TIME:     fr_FR.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.

creating directory /usr/local/pgsql/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 ... Europe/Paris
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:
```

[a] https://www.postgresql.org/docs/current/creating-cluster.html
[b] https://bugzilla.redhat.com/show_bug.cgi?id=1247477#c1

```
pg_ctl -D /usr/local/pgsql/data -l logfile start
```

Avec ces informations, nous pouvons conclure que `initdb` fait les actions suivantes :

- détection de l'utilisateur, de l'encodage et de la locale ;
- création du répertoire PGDATA (`/usr/local/pgsql/data` dans ce cas) ;
- création des sous-répertoires ;
- création et modification des fichiers de configuration ;
- exécution du script bootstrap ;
- synchronisation sur disque ;
- affichage de quelques informations supplémentaires.

De plus, il est possible de changer la méthode d'authentification par défaut avec les paramètres en ligne de commande `--auth`, `--auth-host` et `--auth-local`. Les options `--pwprompt` ou `--pwfile` permettent d'assigner un mot de passe à l'utilisateur **postgres**.

Il est aussi possible d'activer les sommes de contrôle des fichiers de données avec l'option `--data-checksums`. Ces sommes de contrôle sont vérifiées à chaque lecture d'un bloc. Leur activation est chaudement conseillée pour détecter une corruption physique le plus tôt possible. Si votre processeur supporte le jeu d'instruction SSE 4.2 (voir dans `/proc/cpuinfo`), il ne devrait pas y avoir d'impact notable sur les performances. Les journaux générés seront cependant plus nombreux. En version 11, il est possible de vérifier toutes les sommes de contrôle en utilisant l'outil `pg_verify_checksums` (renommé en `pg_checksums` à partir de la version 12). Il n'est pas possible de rajouter les sommes de contrôle sur une instance existante avant la version 12, pensez-y donc dès l'installation.

L'option `--waldir` avait pour nom `--xlogdir` avant la version 10. Cependant, l'option courte n'a pas été modifiée.

L'option `--wal-segsize` n'est disponible qu'à partir de la version 11. Augmenter la valeur par défaut (16 Mo) n'a d'intérêt que pour les très grosses installations générant énormément de journaux pour optimiser leur archivage.

1.2.7 LANCEMENT ET ARRÊT

- Avec le script de l'OS (recommandé) ou `pg_ctl` :

```
# /etc/init.d/postgresql [action]
# service postgresql [action]
# systemctl [action] postgresql
...
$ pg_ctl --pgdata /usr/local/pgsql/data --log logfile [action]
         --mode [smart|fast|immediate]
```

- `[action]` dépend du besoin :
  - `start` / `stop` / `restart`
  - `reload` pour recharger la configuration
  - `init`
  - `status`
  - `promote`
  - `logrotate`
  - `kill`

(Re)démarrage et arrêt :

La méthode recommandée est d'utiliser un script de démarrage adapté à l'OS, (voir plus bas les outils les commandes `systemd` ou celles propres à Debian) , surtout si l'on a installé PostgreSQL par les paquets. Au besoin, un script d'exemple existe dans le répertoire des sources (`contrib/start-scripts/`) pour les distributions Linux et pour les distributions BSD. Ce script est à exécuter en tant qu'utilisateur **root**.

Sinon, il est possible d'exécuter `pg_ctl` avec l'utilisateur créé précédemment.

Les deux méthodes partagent certaines des actions présentées ci-dessus : `start`, `stop` (aux sens évidents) et `reload` pour recharger la configuration.

L'option `--mode` permet de préciser le mode d'arrêt parmi les trois disponibles :

- `smart` : pour vider le cache de PostgreSQL sur disque, interdire de nouvelles connexions et attendre la déconnexion des clients (pgAdmin, pg_dump, psql, etc.) et d'éventuelles sauvegardes ;
- `fast` (par défaut) : pour vider le cache sur disque et déconnecter les clients sans attendre (de ce fait, les transactions en cours sont annulées) ;
- `immediate` : équivalent à un arrêt brutal : tous les processus serveur sont tués et donc, au redémarrage, le serveur devra rejouer ses journaux de transactions.

Rechargement de la configuration :

Pour recharger la configuration après changement du paramétrage, la commande :

```
pg_ctl reload -D /repertoire_pgdata
```

C'est équivalent à cet ordre SQL :

```
SELECT pg_reload_conf() ;
```

Il faut aussi savoir que quelques paramètres nécessitent un redémarrage de PostgreSQL et non un simple rechargement, ils sont indiqués dans les commentaires de `postgresql.conf`.

1.3 INSTALLATION À PARTIR DES PAQUETS LINUX

- Packages Debian
- Packages RPM

Pour une utilisation en environnement de production, il est généralement préférable d'installer les paquets binaires préparés spécialement pour la distribution utilisée. Les paquets sont préparés par des personnes différentes, suivant les recommandations officielles de la distribution. Il y a donc des différences, parfois importantes, entre les paquets.

1.3.1 PAQUETS DEBIAN OFFICIELS

- Nombreux paquets disponibles :
  - serveur, client, contrib, docs
  - extensions, outils
- `apt install postgresql-<version majeure>`
  - installe les binaires
  - crée l'utilisateur **postgres**
  - exécute `initdb`
  - démarre le serveur
- Particularités :
  - wrappers/scripts pour la gestion des différentes instances
  - plusieurs versions majeures installables
  - respect de la FHS

Sous Debian et les versions dérivées (Ubuntu notamment), l'installation de PostgreSQL a été découpée en plusieurs paquets :

- le serveur : `postgresql-<version majeure>` ;
- les clients : `postgresql-client-<version majeure>` ;

- les modules contrib : `postgresql-contrib-<version majeure>` (jusque PostgreSQL 9.6 ; ils sont inclus dans le paquet serveur ensuite) ;
- la documentation : `postgresql-doc-<version majeure>`.

Il existe aussi des paquets pour les outils, les extensions, etc. Certains langages de procédures stockées sont disponibles dans des paquets séparés :

- PL/python dans `postgresql-plpython-<version majeure>`
- PL/perl dans `postgresql-plperl-<version majeure>`
- PL/tcl dans `postgresql-pltcl-<version majeure>`
- etc.

Pour compiler des outils liés à PostgreSQL, il est recommandé d'installer également les bibliothèques de développement qui font partie du paquet `postgresql-server-dev-<version majeure>`.

Le tag `<version majeure>` correspond à la version majeure souhaitée (`9.6` ou `10` par exemple). Cela permet d'installer plusieurs versions majeures sur le même serveur physique ou virtuel.

Les exécutables sont installés dans :

```
/usr/lib/postgresql/<version majeure>/bin
```

Chaque instance porte un nom, qui se retrouve dans le paramètre `cluster_name` et permet d'identifier les processus dans un `ps` ou un `top`. Le nom de la première instance de chaque version majeure est par défaut `main`. Pour chaque instance :

- les données sont dans :

```
/var/lib/postgresql/<version majeure>/<nominstance>
```

- les fichiers de configuration (pas tous ! certains restent dans le répertoire des données) sont dans :

```
/etc/postgresql/<version majeure>/<nominstance>
```

- les traces sont gérées par l'OS sous ce nom :

```
/var/log/postgresql/postgresql-<version majeure>-<nominstance>.log
```

- un fichier PID, la socket d'accès local, et l'espace de travail temporaire des statistiques d'activité figurent dans `/var/run/postgresql`.

Tout ceci vise à respecter le plus possible la norme FHS[3] (*Filesystem Hierarchy Standard*).

[3] http://www.pathname.com/fhs/

Les mainteneurs des paquets Debian ont écrit des scripts pour faciliter la création, la suppression et la gestion de différentes instances sur le même serveur :

- `pg_lsclusters` liste les instances ;
- `pg_createcluster <version majeure> <nom instance>` crée une instance ;
- `pg_dropcluster <version majeure> <nom instance>` détruit une instance ;
- les paramètres par défaut des instances peuvent être centralisés dans `/etc/postgresql-common/createcluster.conf` ;
- la gestion des instances est réalisée avec `pg_ctlcluster` :

```
pg_ctlcluster <version majeure> <nominstance> start|stop|reload|status|promote
```

Quand le paquet serveur est installé, plusieurs opérations sont exécutées : téléchargement du paquet, installation des binaires contenus dans le paquet, création de l'utilisateur **postgres** (s'il n'existe pas déjà), paramétrage d'une première instance nommée `main`, création du répertoire des données, lancement de l'instance. En cas de mise à jour d'un paquet, le serveur PostgreSQL est redémarré après mise à jour des binaires.

Tout ceci explique le grand intérêt de passer par les paquets Debian sur ce type de distribution.

1.3.2 PAQUETS DEBIAN COMMUNAUTAIRES

- La communauté met des paquets Debian à disposition :
  - https://wiki.postgresql.org/wiki/Apt
- Synchrone avec le projet PostgreSQL
- Ajout du dépôt dans `/etc/apt/sources.list.d/pgdg.list` :
  - `deb http://apt.postgresql.org/pub/repos/apt/ buster-pgdg main`

Les paquets de la communauté ont le même contenu que les paquets officiels Debian. La seule différence est qu'ils sont mis à jour plus fréquemment, en liaison directe avec la communauté PostgreSQL.

La distribution Debian préfère des paquets testés et validés, y compris sur des versions assez anciennes, que d'adopter la dernière version dès qu'elle est disponible. Il est donc parfois difficile de mettre à jour avec la dernière version de PostgreSQL. De ce fait, la communauté PostgreSQL met à disposition son propre dépôt de paquets Debian. Elle en assure le maintien et le support. L'équipe de mainteneurs est généralement prévenue trois/quatre jours avant la sortie d'une version pour qu'elle puisse préparer des paquets qui seront disponibles le jour de la sortie officielle.

Les dépôts sont situés sur le serveur `apt.postgresql.org`. La déclaration suivante sera

ajouter dans les fichiers de configuration du gestionnaire `apt`, par exemple pour une distribution Debian 9 (nom de code Stretch) :

```
deb http://apt.postgresql.org/pub/repos/apt/ buster-pgdg main
```

Le nom de code de la distribution peut être obtenu avec la commande :

```
$ lsb_release -cs
```

En combinant les deux commandes précédentes, nous avons la déclaration :

```
# echo "deb http://apt.postgresql.org/pub/repos/apt $(lsb_release -cs)-pgdg main"
```

Enfin, pour finaliser la déclaration, il est nécessaire de récupérer la clé publique du dépôt communautaire :

```
# wget --quiet -O - http://apt.postgresql.org/pub/repos/apt/ACCC4CF8.asc \
    | sudo apt-key add -
# apt update
```

1.3.3 PAQUETS RED HAT OFFICIELS

- Différents paquets disponibles
  - serveur, client, contrib, docs
  - et les extensions, outils
- `yum install postgresqlxx-server`
  - installe les binaires
  - crée l'utilisateur **postgres**
- Opérations manuelles
  - initdb
  - lancement du serveur
- Particularités
  - fichiers de configuration des instances
  - plusieurs versions majeures installables
  - respect du stockage PostgreSQL

Sous Red Hat et les versions dérivées (Rocky Linux, Fedora, CentOS, Scientific Linux par exemple), l'installation de PostgreSQL a été découpée en plusieurs paquets :

- le serveur : `postgresqlXX-server` ;
- les clients : `postgresqlXX` ;
- les modules contrib : `postgresqlXX-contrib` ;
- la documentation : `postgresqlXX-docs`.

où XX est la version majeure (`96` pour 9.6, `10`, `13`...).

Il existe aussi des paquets pour les outils, les extensions, etc. Certains langages de procédures stockées sont disponibles dans des paquets séparés :

- PL/python dans `postgresqlXX-plpython` ;
- PL/perl dans `postgresqlXX-plperl` ;
- PL/tcl dans `postgresqlXX-pltcl` ;
- etc.

Pour compiler des outils liés à PostgreSQL, il est recommandé d'installer également les bibliothèques de développement qui font partie du paquet `postgresqlXX-devel`.

Cela sous-entend qu'il est possible d'installer plusieurs versions majeures sur le même serveur physique ou virtuel. Les exécutables sont installés dans le répertoire `/usr/pgsql-XX/bin`, les traces dans `/var/lib/pgsql/XX/data/log` (utilisation du `logger process` de PostgreSQL), les données dans `/var/lib/pgsql/XX/data`. `data` est le répertoire par défaut des données, mais il est possible de le surcharger. Plutôt que de respecter la norme FHS (`Filesystem Hierarchy Standard`), Red Hat a fait le choix de respecter l'emplacement des fichiers utilisé par défaut par les développeurs PostgreSQL.

Quand le paquet serveur est installé, plusieurs opérations sont exécutées : téléchargement du paquet, installation des binaires contenus dans le paquet, et création de l'utilisateur **postgres** (s'il n'existe pas déjà).

Le répertoire des données n'est pas créé. Cela reste une opération à réaliser par la personne qui a installé PostgreSQL sur le serveur. Cela se fait de deux façons, suivant que le système où est installé PostgreSQL utilise `systemd` ou non. Dans le cas où il est utilisé, un script, nommé `/usr/psql-XX/bin/postgresqlxx-setup`, est mis à disposition pour lancer `initdb` avec les bonnes options. Dans le cas contraire, il faut passer par le script de démarrage, auquel est fournie l'action `initdb` :

```
PGSETUP_INITDB_OPTIONS="--data-checksums" /etc/init.d/postgresql-14 initdb
```

Pour installer plusieurs instances, il est préférable de créer des fichiers de configuration dans le répertoire `/etc/sysconfig/pgsql`. Le nom du fichier de configuration doit correspondre au nom du script de démarrage. La première chose à faire est donc de faire un lien entre le script de démarrage avec un nom permettant de désigner correctement l'instance :

```
# ln -s /etc/init.d/postgresql-XX /etc/init.d/postgresql-serv2
```

Puis, de créer le fichier de configuration avec les paramètres essentiels (généralement `PGDATA`, `PORT`) :

```
# echo >>/etc/sysconfig/pgsql/postgresql-serv2 <<_EOF_
PGDATA=/var/lib/pgsql/XX/serv2
PORT=5433
_EOF_
```

En cas de mise à jour d'un paquet, le serveur PostgreSQL n'est pas redémarré après mise à jour des binaires.

1.3.4 PAQUETS RED HAT COMMUNAUTAIRES

- Préférer les paquets distribués par la communauté :
  - https://yum.postgresql.org/
  - synchrones avec le projet PostgreSQL
- Ajout du dépôt comme paquet RPM

Les versions livrées dans les versions 6 et 7 de Red Hat sont périmées et ne doivent plus être utilisées (versions 8.4 et 9.2 respectivement !). En version 8, même les versions disponibles en AppStream sont un peu en retard. Les dépôts de la communauté sont donc fortement conseillés.

L'installation de la configuration du dépôt de la communauté est très simple. Il suffit de récupérer le paquet RPM de définition du dépôt depuis https://yum.postgresql.org/ et de l'installer, avec les commandes `rpm`, `yum` ou `dnf`. Les commandes peuvent même être générées en fonction des versions sur https://www.postgresql.org/download/linux/redhat/.

Ce dépôt convient également pour les dérivés de Red Hat comme Fedora, CentOS, Rocky Linux…

Il a l'avantage de coller au plus près des versions publiées par la communauté, dans leurs dernières versions, et de fournir nombre d'utilitaires liés à PostgreSQL.

1.4 INSTALLATION SOUS WINDOWS

- Un seul installeur graphique disponible, proposé par EnterpriseDB
- Ou archive des binaires

Le portage de PostgreSQL sous Windows a justifié à lui seul le passage de la branche 7 à la branche 8 du projet. Le système de fichiers NTFS est obligatoire car, contrairement à la VFAT, il gère les liens symboliques (appelés jonctions sous Windows).

L'installateur n'existe plus qu'en version 64 bits depuis PostgreSQL 11.

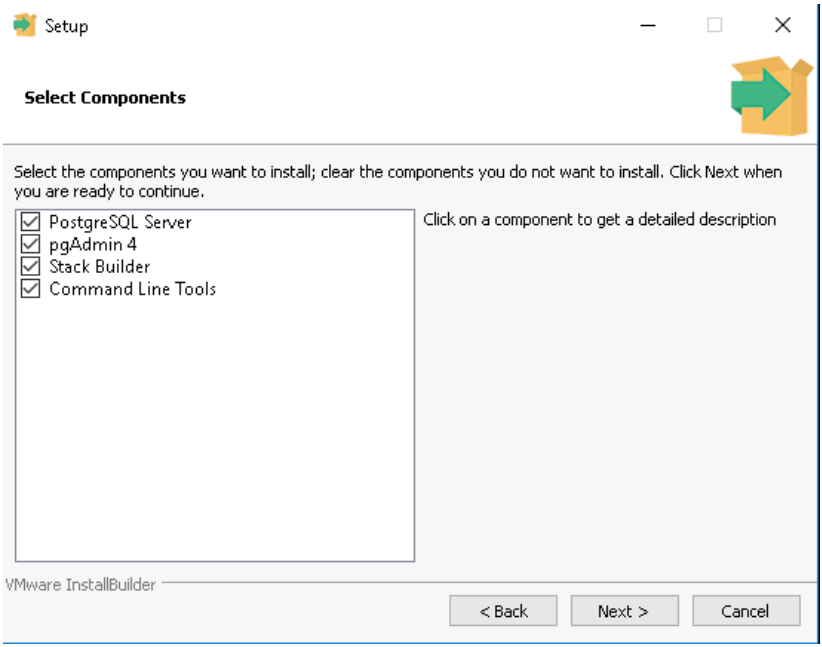
Étant donné la quantité de travail nécessaire pour le développement et la maintenance de l'installeur graphique, la communauté a abandonné l'installeur graphique qu'elle a proposé un temps. EntrepriseDB a continué de proposer gratuitement le sien, pour la version communautaire comme pour leur version payante. D'autres installateurs ont été proposés par d'autres éditeurs.

Il contient le serveur PostgreSQL avec les modules contrib ainsi que pgAdmin 4, et aussi un outil appelé StackBuilder permettant d'installer d'autres outils comme des pilotes JDBC, ODBC, C#, ou PostGIS.

Pour installer PostgreSQL sans installateur ni compilation, EBD propose aussi une archive des binaires compilés[4] .

[4] https://www.enterprisedb.com/download-postgresql-binaries

1.4.1 INSTALLEUR GRAPHIQUE



Son utilisation est tout à fait classique. Il y a plusieurs écrans de saisie d'informations :

- le répertoire d'installation des binaires ;
- le choix des outils (copie d'écran ci-dessus), notamment des outils en ligne de commande (à conserver impérativement), des pilotes et de pgAdmin 4 ;
- le répertoire des données de la première instance ;
- le mot de passe de l'utilisateur **postgres** ;
- le numéro de port ;
- la locale par défaut.

Le répertoire d'installation a une valeur par défaut généralement convenable car il n'existe pas vraiment de raison d'installer les binaires PostgreSQL en dehors du répertoire *Program Files*.

Par contre, le répertoire des données de l'instance PostgreSQL. n'a pas à être dans ce même répertoire *Program Files* ! Il est souvent modifié pour un autre disque que le disque système.

Le numéro de port est par défaut le 5432, sauf si d'autres instances sont déjà installées. Dans ce cas, l'installeur propose un numéro de port non utilisé.

Le mot de passe est celui de l'utilisateur **postgres** au sein de PostgreSQL. En cas de mise à jour, il faut saisir l'ancien mot de passe. Le service lui-même et tous ses processus tourneront avec le compte système générique **NETWORK SERVICE** (Compte de service réseau).

La commande `initdb` est exécutée pour créer le répertoire des données. Un service est ajouté pour lancer automatiquement le serveur au démarrage de Windows.

Un sous-menu du menu *Démarrage* contient le nécessaire pour interagir avec le serveur PostgreSQL, comme une icône pour recharger la configuration et surtout pgAdmin 4, qui se lancera dans un navigateur.

L'outil StackBuilder, lancé dans la foulée, permet de télécharger et d'installer d'autres outils : pilotes pour différents langages (Npgsql pour C#, pgJDBC, psqlODBC), Slony-I, PostGIS… installés dans le répertoire de l'utilisateur en cours.

L'installateur peut être utilisé uniquement en ligne de commande (voir les options avec `--help`).

Cet installeur existe aussi sous macOS X et même Linux (jusqu'en version 10 comprise). Autant il est préférable de passer par les paquets de sa distribution Linux, autant il est recommandé d'utiliser cet installeur sous macOS X.

1.5 PREMIERS RÉGLAGES

- Sécurité
- Configuration minimale
- Démarrage
- Test de connexion

1.5.1 SÉCURITÉ

- Politique d'accès :
  - pour l'utilisateur **postgres** système
  - pour le rôle **postgres**
- Règles d'accès à l'instance dans `pg_hba.conf`

Selon l'environnement et la politique de sécurité interne à l'entreprise, il faut potentiellement initialiser un mot de passe pour l'utilisateur système **postgres** :

```
$ passwd postgres
```

Sans mot de passe, il faudra passer par un système comme `sudo` pour pouvoir exécuter des commandes en tant qu'utilisateur **postgres**, ce qui sera nécessaire au moins au début.

Le fait de savoir qu'un utilisateur existe sur un serveur permet à un utilisateur hostile de tenter de forcer une connexion par force brute. Par exemple, ce billet de blog[5] , montre que l'utilisateur **postgres** est dans le top 10 des logins attaqués.

La meilleure solution pour éviter ce type d'attaque est de ne pas définir de mot de passe pour l'utilisateur OS **postgres** et de se connecter uniquement par des échanges de clés SSH.

Il est conseillé de ne fixer aucun mot de passe pour l'utilisateur système. Il en va de même pour le rôle **postgres** dans l'instance. Une fois connecté au système, nous pourrons utiliser le mode d'authentification local `peer` pour nous connecter au rôle **postgres**. Ce mode permet de limiter la surface d'attaque sur son instance.

En cas de besoin d'accès distant en mode superutilisateur, il sera possible de créer des rôles supplémentaires avec des droits superutilisateur. Ces noms ne doivent pas être facile à deviner par de potentiels attaquants. Il faut donc éviter les rôles **admin** ou **root**.

Si vous avez besoin de créer des mots de passe, ils doivent bien sûr être longs et complexes (par exemple en les générant avec les utilitaires `pwgen` ou `apg`).

Si vous avez utilisé l'installeur proposé par EnterpriseDB, l'utilisateur système et le rôle PostgreSQL ont déjà un mot de passe, celui demandé par l'installeur. Il n'est donc pas nécessaire de leur en configurer un autre.

Enfin, il est important de vérifier les règles d'accès au serveur contenues dans le fichier `pg_hba.conf`. Ces règles définissent les accès à l'instance en se basant sur plusieurs paramètres : utilisation du réseau ou du socket fichier, en SSL ou non, depuis quel réseau, en utilisant quel rôle, pour quelle base de données et avec quelle méthode d'authentification.

[5] https://blog.sucuri.net/2013/07/ssh-brute-force-the-10-year-old-attack-that-still-persists.html

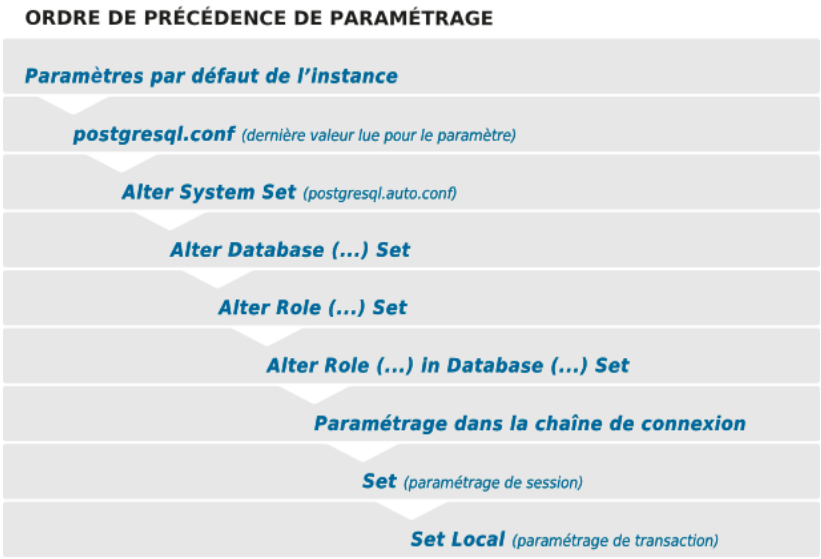
1.5.2 CONFIGURATION MINIMALE

- Fichier `postgresql.conf`
- Configuration du moteur
- Plus de 300 paramètres
- Quelques paramètres essentiels

La configuration du moteur se fait via un seul fichier, le fichier `postgresql.conf`. Il se trouve généralement dans le répertoire des données du serveur PostgreSQL. Sous certaines distributions (Debian et affiliés principalement), il est déplacé dans `/etc/postgresql/`.

Ce fichier contient beaucoup de paramètres, plus de 300, mais seuls quelques-uns sont essentiels à connaître pour avoir une instance fiable et performante.

1.5.3 CONFIGURATION PRÉCÉDENCE DES PARAMÈTRES



PostgreSQL offre une certaine granularité dans sa configuration, ainsi certains paramètres

peuvent être surchargés par rapport au fichier `postgresql.conf`. Il est utile de connaître l'ordre de précédence. Par exemple, un utilisateur peut spécifier un paramètre dans sa session avec l'ordre `SET`, celui-ci sera prioritaire par rapport à la configuration présente dans le fichier `postgresql.conf`.

1.5.4 CONFIGURATION DES CONNEXIONS

- `listen_addresses = '*'`
- `port = 5432`
- `max_connections = 100`
  - compromis nombre de requêtes actives/nombre de CPU/complexité des requêtes
  - ne pas monter trop haut
  - sinon, penser au pooling
- `password_encryption` = `scram-sha-256` (v10+)

Par défaut, le serveur installé n'écoute pas sur les interfaces réseaux externes. Pour autoriser les clients externes à se connecter à PostgreSQL, il faut modifier le paramètre `listen_addresses`.

La valeur `*` est un joker indiquant que PostgreSQL doit écouter sur toutes les interfaces réseaux disponibles au moment où il est lancé. Il est aussi possible d'indiquer les interfaces, une à une, en les séparant avec des virgules. Cette méthode est intéressante lorsqu'on veut éviter que l'instance écoute sur une interface donnée. Par prudence il est possible de se limiter aux interfaces destinées à être utilisées, même si cela est un peu redondant avec la configuration de `pg_hba.conf` :

```
listen_addresses = 'localhost, 10.1.123.123'
```

Le port par défaut des connexions TCP/IP est le `5432`. Il est possible de le modifier. Cela n'a réellement un intérêt que si vous voulez exécuter plusieurs serveurs PostgreSQL sur le même serveur (physique ou virtuel). En effet, plusieurs serveurs sur une même machine ne peuvent pas écouter sur le même port. PostgreSQL n'écoute que sur cet unique port.

Une instance PostgreSQL n'écoute jamais que sur ce seul port, et tous les clients se connectent dessus. Il n'existe pas de notion de *listener* ou d'outil de redirection comme sur d'autres bases de données concurrentes, du moins sans outil supplémentaire.

Ne confondez pas la connexion à `localhost` (soit `::1` ou `127.0.0.1`) et celle dite `local`, passant par les *sockets* de l'OS (par défaut `/var/run/postgresql/.s.PGSQL.5432` sur les distributions les plus courantes).

Le nombre de connexions simultanées est limité par le paramètre `max_connections`. Dès que ce nombre est atteint, les connexions suivantes sont refusées jusqu'à ce qu'un utilisateur connecté se déconnecte. La valeur par défaut de `100` est généralement suffisante. Il peut être intéressant de la diminuer pour monter `work_mem` et autoriser plus de mémoire de tri. Il est possible de l'augmenter pour qu'un plus grand nombre d'utilisateurs puisse se connecter en même temps.

Il faut surtout savoir qu'à chaque connexion se voit associé un processus sur le serveur, qui n'est vraiment actif qu'à l'exécution d'une requête. Il s'agit aussi d'arbitrer entre le nombre de requêtes à exécuter à un instant T, le nombre de CPU disponibles, la complexité des requêtes, et le nombre de processus que peut gérer l'OS.

Cela est compliqué par le fait qu'une même requête peut mobiliser plusieurs processeurs si elle est parallélisée (à partir de PostgreSQL 9.6), et que, selon la requête, elle sera limitée par le CPU ou la bande passante des disques.

Enfin, même si une connexion inactive ne consomme pas de CPU, elle fera dépenser du CPU à une connexion active. En effet, une connexion active va générer assez fréquemment ce qu'on appelle un snapshot (ou une image) de l'état des transactions de la base. La durée de création de ce snapshot dépend principalement du nombre de connexions, actives ou non, sur le serveur. Donc une connexion active consommera plus de CPU s'il y a 399 autres connexions, actives ou non, que s'il y a 9 connexions, actives ou non. Ce comportement devrait être partiellement corrigé avec la version 14.

Au niveau mémoire, un processus consomme par défaut 2 Mo de mémoire vive. Cette consommation peut augmenter suivant son activité.

Intercaler un « pooler » comme pgBouncer entre les clients et l'instance peut se justifier dans certains cas :

- connexions/déconnexions très fréquentes (la connexion a un coût) ;
- centaines, voire milliers, de connexions généralement inactives.

À partir de la version 10 et avant la version 14, le paramètre `password_encryption` est à modifier dès l'installation. Il définit l'algorithme de chiffrement utilisé pour le stockage des mots de passe. La valeur `scram-sha-256` permettra d'utiliser la nouvelle norme, plus sécurisée que l'ancien `md5`. Ce n'est plus nécessaire à partir de la version 14 car c'est la valeur par défaut. Avant toute modification, vérifiez quand même que vos outils clients sont compatibles. Au besoin, vous pouvez revenir à `md5` pour un utilisateur donné.

1.5.5 CONFIGURATION DES RESSOURCES MÉMOIRE

Mémoire partagée :

- `shared_buffers`

Mémoire des processus :

- `work_mem`
- `hash_mem_multiplier`
- `maintenance_work_mem`

Chaque fois que PostgreSQL a besoin de lire ou d'écrire des données, il les place d'abord dans son cache interne. Ce cache ne sert qu'à ça : stocker des blocs disques qui sont accessibles à tous les processus PostgreSQL, ce qui permet d'éviter de trop fréquents accès disques car ces accès sont lents. La taille de ce cache dépend d'un paramètre appelé `shared_buffers`.

Pour dimensionner `shared_buffers` sur un serveur dédié à PostgreSQL, la documentation officielle[a] donne 25 % de la mémoire vive totale comme un bon point de départ et déconseille de dépasser 40 %, car le cache du système d'exploitation est aussi utilisé.

Sur une machine de 32 Go de RAM, cela donne donc :

```
shared_buffers=8GB
```

Le défaut de 128 Mo n'est pas adapté à un serveur sur une machine récente.

À cause du coût de la gestion de cette mémoire, surtout avec de nombreux processeurs ou de nombreux clients, une règle conservatrice peut être de ne pas dépasser 8 ou 10 Go, surtout sur les versions les moins récentes de PostgreSQL. Jusqu'en 9.6, sous Windows, il était même conseillé de ne pas dépasser 512 Mo.

Suivant les cas, une valeur inférieure ou supérieure à 25 % sera encore meilleure pour les performances, mais il faudra tester avec votre charge (en lecture, en écriture, et avec le bon nombre de clients).

Attention : une valeur élevée de `shared_buffers` (au-delà de 8 Go) nécessite de paramétrer finement le système d'exploitation (*Huge Pages* notamment) et d'autres paramètres comme `max_wal_size`, et de s'assurer qu'il restera de la mémoire pour le reste des opérations (tri…).

`work_mem` et `maintenance_work_mem` sont des paramètres mémoires utilisés par chaque processus. `work_mem` est la taille de la mémoire allouée aux opérations de tri, alors que `maintenance_work_mem` est la taille de la mémoire pour les `VACUUM`, `CREATE INDEX` et ajouts de clé étrangère. La valeur de `work_mem` dépend beaucoup de la mémoire disponible, des

[a] https://www.postgresql.org/docs/current/runtime-config-resource.html

usages et du nombre de connexions, et vaut souvent quelques dizaines de Mo, parfois beaucoup plus. `maintenance_work_mem` vaut souvent 256 Mo à 1 Go sur les machines récentes, parfois plus.

À partir de PostgreSQL 13, `hash_mem_multiplier` permet de configurer un coefficient multiplicateur qui, associé au paramètre `work_mem`, permet de calculer la mémoire maximale allouée pour des opérations de hachage lors d'agrégations. La valeur par défaut 1 correspond au comportement des versions jusqu'à la 12.

La mémoire allouée grâce à `work_mem`, `maintenance_work_mem` et `hash_mem_multiplier` est distincte de celle définie par le paramètre `shared_buffers`, et elle peut être allouée pour chaque étape de chaque requête. Donc, si votre `max_connections` est important, ces valeurs (en fait, surtout `work_mem`) devront être faibles. Alors que si votre `max_connections` est faible, ces valeurs pourront être augmentées.

1.5.6 CONFIGURATION DES JOURNAUX DE TRANSACTIONS 1/2

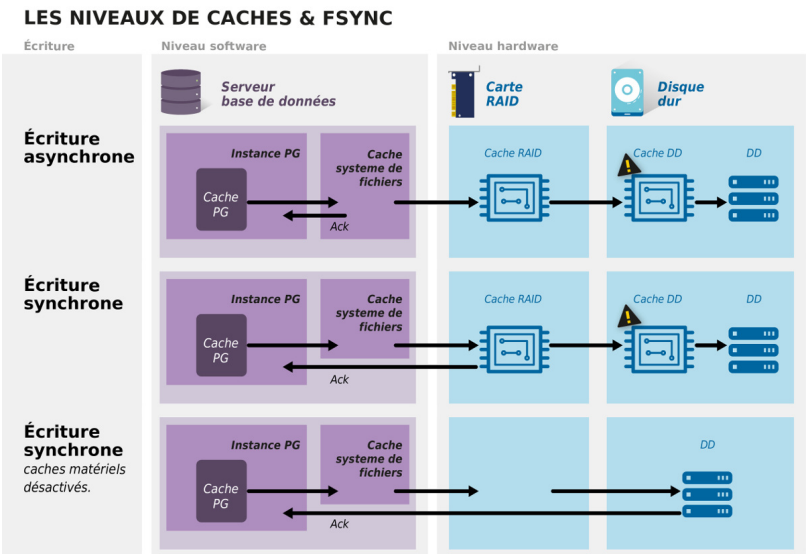
- Paramètre : `fsync`
- Laisser à `on` si vous tenez à vos données

À chaque fois qu'une transaction est validée (`COMMIT`), PostgreSQL écrit les modifications qu'elle a générées dans les journaux de transactions.

Afin de garantir la durabilité, PostgreSQL effectue des écritures synchrones des journaux de transaction, donc une écriture physique des données sur le disque. Cela a un coût important sur les performances en écritures s'il y a de nombreuses transactions mais c'est le prix de la sécurité.

Le paramètre `fsync` permet de désactiver l'envoi de l'ordre de synchronisation au système d'exploitation. Ce paramètre **doit** rester à `on` en production. Dans le cas contraire, un arrêt brutal de la machine peut mener à la perte des journaux non encore enregistrés et à la corruption de l'instance. D'autres paramètres et techniques existent pour gagner en performance (et notamment si certaines données peuvent être perdues) sans pour autant risquer de corrompre l'instance.

1.5.7 CONFIGURATION DES JOURNAUX DE TRANSACTIONS 2/2



Une écriture peut être soit synchrone soit asynchrone. Pour comprendre ce mécanisme, nous allons simplifier le cheminement de l'écriture d'un bloc :

- Dans le cas d'une écriture **asynchrone** : Un processus qui modifie un fichier écrit en fait d'abord dans le cache du système de fichiers du système d'exploitation (OS), cache situé en RAM (mémoire volatile). L'OS confirme tout de suite au processus que l'écriture a été réalisée pour lui rendre la main au plus vite : il y a donc un gain en performance important. Cependant, le bloc ne sera écrit sur disque que plus tard afin notamment de grouper les demandes d'écritures des autres processus, et de réduire les déplacements des têtes de lecture/écriture des disques, qui sont des opérations coûteuses en temps. Entre la confirmation de l'écriture et l'écriture réelle sur les disques, il peut se passer un certain délai : si une panne survient durant celui-ci, les données soi-disant écrites seront perdues, car pas encore physiquement sur le disque.

- Dans le cas d'une écriture **synchrone** : Un processus écrit dans le cache du système d'exploitation, puis demande explicitement à l'OS d'effectuer la synchronisation (écriture physique) sur disque. Les blocs sont donc écrits sur les disques immédiatement et le processus n'a la confirmation de l'écriture qu'une fois cela fait.

> Il attendra donc pendant la durée de cette opération, mais il aura la garantie que la donnée est bien présente physiquement sur les disques. Cette synchronisation est très coûteuse et lente (encore plus avec un disque dur classique et ses têtes de disques à déplacer).

Un phénomène équivalent peut se produire à nouveau au niveau matériel (hors du contrôle de l'OS) : pour gagner en performance, les constructeurs ont rajouté un système de cache au sein des cartes RAID. L'OS (et donc le processus qui écrit) a donc confirmation de l'écriture dès que la donnée est présente dans ce cache, alors qu'elle n'est pas encore écrite sur disque. Afin d'éviter la perte de donnée en cas de panne électrique, ce cache est secouru par une batterie qui laissera le temps d'écrire le contenu du cache. Vérifiez qu'elle est bien présente sur vos disques et vos cartes contrôleur RAID.

1.5.8 CONFIGURATION DES TRACES

- Selon système/distribution :
  - `log_destination`
  - `logging_collector`
  - emplacement et nom différent pour `postgresql-????.log`
- `log_line_prefix` à compléter :
  - `log_line_prefix = '%t [%p]: user=%u,db=%d,app=%a,client=%h '`
- `lc_messages` = `C` (anglais)

PostgreSQL dispose de plusieurs moyens pour enregistrer les traces : soit il les envoie sur la sortie des erreurs (`stderr` et `csvlog`), soit il les envoie à syslog (`syslog`, seulement sous Unix), soit il les envoie au journal des événements (`eventlog`, sous Windows uniquement). Dans le cas où les traces sont envoyées sur la sortie des erreurs, il peut récupérer les messages via un démon appelé *logger process* qui va enregistrer les messages dans des fichiers. Ce démon s'active en configurant le paramètre `logging_collector` à `on`.

Tout cela est configuré par défaut différemment selon le système et la distribution. Red Hat active `logging_collector` et PostgreSQL dépose ses traces dans des fichiers journaliers `$PGDATA/log/postgresql-<jour de la semaine>.log`. Debian utilise `stderr` sans autre paramétrage et c'est le système qui dépose les traces dans `/var/log/postgresql/postgresql-VERSION-nominstance.log`. Les deux variantes fonctionnent. En fonction des habitudes et contraintes locales, il est possible de préférer et d'activer l'une ou l'autre.

L'entête de chaque ligne des traces doit contenir au moins la date et l'heure exacte (`%t` ou `%m` suivant la précision désirée) : des traces sans date et heure ne servent à rien. Des

entêtes complets sont suggérés par la documentation de l'analyseur de log pgBadger :

```
log_line_prefix = '%t [%p]: [%l-1] db=%d,user=%u,app=%a,client=%h '
```

Beaucoup d'utilisateurs français récupèrent les traces de PostgreSQL en français. Bien que cela semble une bonne idée au départ, cela se révèle être souvent un problème. Non pas à cause de la qualité de la traduction, mais plutôt parce que les outils de traitement des traces fonctionnent uniquement avec des traces en anglais. Même un outil comme pgBadger, pourtant écrit par un Français, ne sait pas interpréter des traces en français. De plus, la moindre recherche sur Internet ramènera plus de liens si le message est en anglais. Positionnez donc `lc_messages` à `C`.

1.5.9 CONFIGURATION DES DÉMONS

Laisser ces deux paramètres à `on` :

- `autovacuum`
- `stats collector`

En dehors du *logger process*, PostgreSQL dispose d'autres démons.

L'autovacuum joue un rôle important pour de bonnes performances : il empêche une fragmentation excessive des tables et index, et met à jour les statistiques sur les données (qui servent à l'optimiseur de requêtes).

Le collecteur de statistiques sur l'activité permet le bon fonctionnement de l'autovacuum et donne de nombreuses informations importantes à l'administrateur de bases de données.

Ces deux démons devraient toujours être activés.

1.5.10 SE FACILITER LA VIE

- Création automatique de configuration
  - pgtune et https://pgtune.leopard.in.ua/
  - http://pgconfigurator.cybertec.at/
- Documentation et analyse de configuration
  - https://postgresqlco.nf

pgtune existe en plusieurs versions. La version en ligne de commande va détecter automatiquement le nombre de CPU et la quantité de RAM, alors que la version web nécessitera que ces informations soient saisies. Suivant le type d'utilisation, pgtune proposera une

configuration adaptée. Cette configuration n'est évidemment pas forcément optimale par rapport à vos applications, tout simplement parce qu'il ne connaît que les ressources et le type d'utilisation, mais c'est généralement un bon point de départ.

pgconfigurator est un outil plus récent, un peu plus graphique, mais il remplit exactement le même but que pgtune.

Enfin, le site postgresql.co.nf[6] est un peu particulier. C'est en quelque sorte une encyclopédie sur les paramètres de PostgreSQL, mais il est aussi possible de lui faire analyser une configuration. Après analyse, des informations supplémentaires seront affichées pour améliorer cette configuration, que ce soit pour la stabilité du serveur comme pour ses performances.

1.6 MISE À JOUR

- Recommandations
- Mise à jour mineure
- Mise à jour majeure

1.6.1 RECOMMANDATIONS

- Les *Release Notes*
- Intégrité des données
- Bien redémarrer le serveur !

Chaque nouvelle version de PostgreSQL est accompagnée d'une note expliquant les améliorations, les corrections et les innovations apportées par cette version, qu'elle soit majeure ou mineure. Ces notes contiennent toujours une section dédiée aux mises à jour dans laquelle se trouvent des conseils essentiels.

Les *Releases Notes* sont présentes dans l'annexe E de la documentation officielle[7] .

Les données de votre instance PostgreSQL sont toujours compatibles d'une version mineure à l'autre. Ainsi, les mises à jour vers une version mineure supérieure peuvent se faire sans migration de données, sauf cas exceptionnel qui serait alors précisé dans les notes de version. Par exemple, de la 9.6.1 à la 9.6.2, il est nécessaire de reconstruire les index construits en mode concurrent pour éviter certaines corruptions. À partir de la

[6] https://postgresqlco.nf
[7] https://docs.postgresql.fr/current/release.html

10.3, `pg_dump` impose des noms d'objets qualifiés pour des raisons de sécurité, ce qui a posé problème pour certains réimports.

Pensez éventuellement à faire une sauvegarde préalable par sécurité.

À contrario, si la mise à jour consiste en un changement de version majeure (par exemple, de la 9.4 à la 9.6), alors il est nécessaire de s'assurer que les données seront transférées correctement sans incompatibilité. Là encore, il est important de lire les *Releases Notes* **avant** la mise à jour.

Le site https://why-upgrade.depesz.com/, basé sur les release notes, permet de compiler les différences entre plusieurs versions de PostgreSQL.

Dans tous les cas, pensez à bien redémarrer le serveur. Mettre à jour les binaires ne suffit pas.

1.6.2 MISE À JOUR MINEURE

- Méthode
  - arrêter PostgreSQL
  - mettre à jour les binaires
  - redémarrer PostgreSQL
- Pas besoin de s'occuper des données, sauf cas exceptionnel
  - bien lire les *Release Notes* pour s'en assurer

Faire une mise à jour mineure est simple et rapide.

La première action est de lire les *Release Notes* pour s'assurer qu'il n'y a pas à se préoccuper des données. C'est généralement le cas mais il est préférable de s'en assurer avant qu'il ne soit trop tard.

La deuxième action est de faire la mise à jour. Tout dépend de la façon dont PostgreSQL a été installé :

- par compilation, il suffit de remplacer les anciens binaires par les nouveaux ;
- par paquets précompilés, il suffit d'utiliser le système de paquets (`apt` sur Debian et affiliés, `yum` ou `dnf` sur Red Hat et affiliés) ;
- par l'installeur graphique, en le ré-exécutant.

Ceci fait, un redémarrage du serveur est nécessaire. Il est intéressant de noter que les paquets Debian s'occupent directement de cette opération. Il n'est donc pas nécessaire de le refaire.

1.6.3 MISE À JOUR MAJEURE

- Bien lire les *Release Notes*
- Bien tester l'application avec la nouvelle version
  - rechercher les régressions en terme de fonctionnalités et de performances
  - penser aux extensions et aux outils
- Pour mettre à jour
  - mise à jour des binaires
  - et mise à jour/traitement des fichiers de données
- 3 méthodes
  - dump/restore
  - réplication logique, externe (Slony) ou interne
  - `pg_upgrade`

Faire une mise à jour majeure est une opération complexe à préparer prudemment.

La première action là-aussi est de lire les *Release Notes* pour bien prendre en compte les régressions potentielles en terme de fonctionnalités et/ou de performances. Cela n'arrive presque jamais mais c'est possible malgré toutes les précautions mises en place.

La deuxième action est de mettre en place un serveur de tests où se trouve la nouvelle version de PostgreSQL avec les données de production. Ce serveur sert à tester PostgreSQL mais aussi, et même surtout, l'application. Le but est de vérifier encore une fois les régressions possibles.

N'oubliez pas de tester les extensions non officielles, voire développées en interne, que vous avez installées. Elles sont souvent moins bien testées.

N'oubliez pas non plus de tester les outils d'administration, de monitoring, de modélisation. Ils nécessitent souvent une mise à jour pour être compatibles avec la nouvelle version installée.

Une fois que les tests sont concluants, arrive le moment de la mise en production. C'est une étape qui peut être longue car les fichiers de données doivent être traités. Il existe plusieurs méthodes que nous détaillerons après.

1.6.4 MISE À JOUR MAJEURE PAR DUMP/RESTORE

- Méthode historique
- Simple et sans risque
  - mais d'autant plus longue que le volume de données est important
- Outils `pg_dump` (ou `pg_dumpall`) et `pg_restore`

Il s'agit de la méthode la plus ancienne et la plus sûre. L'idée est de sauvegarder l'ancienne version avec l'outil de sauvegarde de la nouvelle version. `pg_dumpall` peut suffire, mais `pg_dump` est malgré tout recommandé. Le problème de lenteur vient surtout de la restauration. `pg_restore` est un outil assez lent pour des volumétries importantes. Il convient donc de sélectionner cette solution si le volume de données n'est pas conséquent (pas plus d'une centaine de Go) ou si les autres méthodes ne sont pas possibles. Cependant, il est possible d'accélérer la restauration en utilisant la parallélisation (option `--jobs`). Ceci n'est possible que si la sauvegarde a été faite avec `pg_dump -Fd` ou `-Fc`. Il est à noter que cette sauvegarde peut elle aussi être parallélisée (option `--jobs` là encore).

1.6.5 MISE À JOUR MAJEURE PAR SLONY

- Nécessite d'utiliser l'outil de réplication Slony
- Permet un retour en arrière immédiat sans perte de données

La méthode Slony est certainement la méthode la plus compliquée. C'est aussi une méthode qui permet un retour arrière vers l'ancienne version sans perte de données.

L'idée est d'installer la nouvelle version de PostgreSQL normalement, sur le même serveur ou sur un autre serveur. Il faut installer Slony sur l'ancienne et la nouvelle instance, et déclarer la réplication de l'ancienne instance vers la nouvelle. Les utilisateurs peuvent continuer à travailler pendant le transfert initial des données. Ils n'auront pas de blocages, tout au plus une perte de performances dues à la lecture et à l'envoi des données vers le nouveau serveur. Une fois le transfert initial réalisé, les données modifiées entre temps sont transférées vers le nouveau serveur.

Une fois arrivé à la synchronisation des deux serveurs, il ne reste plus qu'à déclencher un *switchover*. La réplication aura lieu ensuite entre le nouveau serveur et l'ancien serveur, ce qui permet un retour en arrière sans perte de données. Une fois acté que le nouveau serveur donne pleine satisfaction, il suffit de désinstaller Slony des deux côtés.

1.6.6 MISE À JOUR MAJEURE PAR RÉPLICATION LOGIQUE

- Possible entre versions 10 et supérieures
- Remplace Slony, Bucardo...
- Bascule très rapide
- Et retour possible

La réplication logique rend possible une migration entre deux instances de version majeure différente avec une indisponibilité très courte.

La réplication logique n'est disponible en natif qu'à partir de la version 10, la base à migrer doit donc être en version 10 ou supérieure.

Le même principe que les outils de réplication par trigger comme Slony ou Bucardo est utilisé, mais plus simplement et avec les outils du moteur. Le principe est de répliquer une base à l'identique alors que la production tourne. Des clés primaires sur chaque table sont souhaitables mais pas forcément obligatoires.

Lors de la bascule, il suffit d'attendre que les dernières données soient répliquées, ce qui peut être très rapide, et de connecter les applications au nouveau serveur. La réplication peut alors être inversée pour garder l'ancienne production synchrone, permettant de rebasculer dessus en cas de problème sans perdre les données modifiées depuis la bascule.

1.6.7 MISE À JOUR MAJEURE PAR PG_UPGRADE

- `pg_upgrade` développé par la communauté depuis la version 8.4
  - et fourni avec PostgreSQL
- Prend comme prérequis que le format de stockage des fichiers de données utilisateurs ne change pas entre versions
- Nécessite les deux versions sur le même serveur

`pg_upgrade` est certainement l'outil le plus rapide pour une mise à jour majeure. Grossièrement, son fonctionnement est le suivant. Il récupère la déclaration des objets sur l'ancienne instance avec un `pg_dump` du schéma de chaque base et de chaque objet global. Il intègre la déclaration des objets dans la nouvelle instance. Il fait un ensemble de traitement sur les identifiants d'objets et de transactions. Puis, il copie les fichiers de données de l'ancienne instance vers la nouvelle instance. La copie est l'opération la plus longue mais comme il n'est pas nécessaire de reconstruire les index et de vérifier les contraintes, cette opération est bien plus rapide que la restauration d'une sauvegarde style `pg_dump`. Pour aller plus rapidement, il est aussi possible de créer des liens physiques à la place de la copie des fichiers. Ceci fait, la migration est terminée.

En 2010, Stefan Kaltenbrunner et Bruce Momjian avaient mesuré qu'une base de 150 Go mettait 5 heures à être mise à jour avec la méthode historique (sauvegarde/restauration). Elle mettait 44 minutes en mode copie et 42 secondes en mode lien lors de l'utilisation de `pg_upgrade`.

Vu ses performances, ce serait certainement l'outil à privilégier. Cependant, c'est un outil très complexe et quelques bugs particulièrement méchants ont terni sa réputation. Notre recommandation est de bien tester la mise à jour avant de le faire en production, et uniquement sur des bases suffisamment volumineuses permettant de justifier l'utilisation de cet outil.

1.6.8 MISE À JOUR DE L'OS

Si vous migrez aussi l'OS ou déplacez les fichiers d'une instance :

- compatibilité architecture
- compatibilité librairies
  - réindexation parfois nécessaire
  - ex : Debian 10 et glibc 2.28

Un projet de migration PostgreSQL est souvent l'occasion de mettre à jour le système d'exploitation. Vous pouvez également en profiter pour déplacer l'instance sur un autre serveur à l'OS plus récent en copiant (à froid) le PGDATA.

Il faut bien sûr que l'architecture physique (32/64 bits, *big/little indian*) reste la même. Cependant, même entre deux versions de la même distribution, certains composants du système d'exploitation peuvent avoir une influence, à commencer par la `glibc`. Cette dernière définit l'ordre des caractères, ce qui se retrouve dans les index. Une incompatibilité entre deux versions sur ce point oblige donc à reconstruire les index, sous peine d'incohérence avec les fonctions de comparaison sur le nouveau système et de corruption à l'écriture.

Daniel Vérité détaille sur son blog[8] le problème pour les mises à jour entre Debian 9 et 10, à cause de la mise à jour de la `glibc`. L'utilisation des collations ICU[9] dans les index contourne le problème mais elles sont encore peu répandues.

Ce problème ne touche bien sûr pas les migrations ou les restaurations avec `pg_dump`/`pg_restore` : les données sont alors transmises de manière logique, indépendamment des caractéristiques physiques des instances source et cible, et les index sont systématiquement reconstruits sur la machine cible.

[8] https://blog-postgresql.verite.pro/2018/08/30/glibc-upgrade.html
[9] https://blog-postgresql.verite.pro/2018/07/27/icu_ext.html

1.7 CONCLUSION

- L'installation est simple....
- ... mais elle doit être soigneusement préparée
- Préférer les paquets officiels
- Attention aux données lors d'une mise à jour !

1.7.1 POUR ALLER PLUS LOIN

- Documentation officielle, chapitre `Installation`
- Documentation Dalibo, pour l'installation sur Windows

Vous pouvez retrouver la documentation en ligne sur https://docs.postgresql.fr/current/installation.html.

La documentation de Dalibo pour l'installation de PostgreSQL sur Windows est disponible sur https://public.dalibo.com/archives/etudes/installer_postgresql_9.0_sous_windows.pdf.

1.7.2 QUESTIONS

N'hésitez pas, c'est le moment !

1.8 QUIZ

https://dali.bo/b_quiz

1.9 TRAVAUX PRATIQUES

1.9.1 INSTALLATION À PARTIR DES SOURCES (OPTIONNEL)

But : Installer PostgreSQL à partir du code source

Note : Pour éviter tout problème lié au positionnement des variables d'environnement dans les exercices suivants, l'installation depuis les sources se fera avec un utilisateur dédié, différent de l'utilisateur utilisé par l'installation depuis les paquets de la distribution.

Outils de compilation

> Installer les outils de compilation suivants, si ce n'est déjà fait.

Sous Rocky Linux 8, il faudra utiliser `dnf` :

```
dnf -y group install "Development Tools"
dnf -y install readline-devel openssl-devel wget
```

Sous CentOS 7 :

```
yum -y install bison-devel readline-devel zlib-devel openssl-devel wget
yum -y groupinstall 'Development Tools'
```

Sous Debian ou Ubuntu :

```
apt install -y build-essential libreadline-dev zlib1g-dev flex bison \
  libxml2-dev libxslt-dev libssl-dev
```

> Créer l'utilisateur système **srcpostgres** avec `/opt/pgsql` pour répertoire `HOME`.

> Se connecter en tant que l'utilisateur **srcpostgres**.

Téléchargement

> Consulter le site officiel du projet et relever la dernière version de PostgreSQL.
> Télécharger l'archive des fichiers sources de la dernière version stable et les placer dans `/opt/pgsql/src`.

Compilation et installation

> L'installation des binaires compilés se fera dans `/opt/pgsql/14/`.
> Configurer en conséquence l'environnement de compilation (`./configure`).
> Compiler PostgreSQL.

> Installer les fichiers obtenus.

> Où se trouvent les binaires installés de PostgreSQL ?

Configurer le système

> Ajouter les variables d'environnement `PATH` et `LD_LIBRARY_PATH` au `~srcpostgres/.bash_profile` de l'utilisateur **srcpostgres** pour accéder facilement à ces binaires.

Création d'une instance

> Avec `initdb`, initialiser une instance dans `/opt/pgsql/14/data` en spécifiant **postgres** comme nom de super-utilisateur, et en activant les sommes de contrôle.

> Démarrer l'instance.

> Tenter une première connexion avec `psql`.
> Pourquoi cela échoue-t-il ?

> Se connecter en tant qu'utilisateur **postgres**. Ressortir.

> Dans `.bash_profile`, configurer la variable d'environnement `PGUSER`.

Première base

> Créer une première base de donnée nommée `test`.

> Se connecter à la base `test` et créer quelques tables.

Arrêt

> Arrêter cette instance.

1.9.2 INSTALLATION DEPUIS LES PAQUETS BINAIRES DU PGDG

But : Installer PostgreSQL à partir des paquets communautaires
Cette instance servira aux TP suivants.

Pré-installation

> Quelle commande permet d'installer les paquets binaires de PostgreSQL ?

> Quelle version est packagée ?

> Quels paquets devront également être installés ?

Installation

> Installer le dépôt.

> Désactiver le module d'installation pour la version PostgreSQL de la distribution.

> Installer les paquets de PostgreSQL14 : serveur, client, contribs.

> Quel est le chemin des binaires ?

Création de la première instance

> Créer une première instance avec les outils de la famille Red Hat en activant les sommes de contrôle (*checksums*).

> Vérifier ce qui a été fait dans le journal `initdb.log`.

Démarrage

> Démarrer l'instance.

> Activer le démarrage de l'instance au démarrage de la machine.

> Où sont les fichiers de données (`PGDATA`), et les traces de l'instance ?

Configuration

> Vérifier la configuration par défaut de PostgreSQL. Est-ce que le serveur écoute sur le réseau ?

> Quel est l'utilisateur sous lequel tourne l'instance ?

Connexion

> En tant que **root**, tenter une connexion avec `psql`.

> En tant que **postgres**, tenter une connexion avec `psql`.
> Quitter.

> À quelle base se connecte-t-on par défaut ?

> Créer une première base de données et y créer des tables.

1.10 TRAVAUX PRATIQUES (SOLUTIONS)

1.10.1 INSTALLATION À PARTIR DES SOURCES (OPTIONNEL)

Outils de compilation

> Installer les outils de compilation suivants, si ce n'est déjà fait.

Ces actions doivent être effectuées en tant qu'utilisateur privilégié (soit directement en tant que **root**, soit en utilisant la commande `sudo`).

Sous Rocky Linux 8, il faudra utiliser `dnf` :

```
dnf -y group install "Development Tools"
dnf -y install readline-devel openssl-devel wget
```

Sous CentOS 7 :

```
yum -y install bison-devel readline-devel zlib-devel openssl-devel wget
yum -y groupinstall 'Development Tools'
```

Sous Debian ou Ubuntu :

```
apt install -y build-essential libreadline-dev zlib1g-dev flex bison \
  libxml2-dev libxslt-dev libssl-dev
```

Une fois ces outils installés, tout ce qui suit devrait fonctionner sur toute version de Linux.

> Créer l'utilisateur système **srcpostgres** avec `/opt/pgsql` pour répertoire `HOME`.

En tant que **root** :

```
useradd --home-dir /opt/pgsql --system --create-home srcpostgres
usermod --shell /bin/bash srcpostgres
```

> Se connecter en tant que l'utilisateur **srcpostgres**.

Se connecter en tant qu'utilisateur **srcpostgres** :

```
su - srcpostgres
```

Téléchargement

> Consulter le site officiel du projet et relever la dernière version de PostgreSQL.
> Télécharger l'archive des fichiers sources de la dernière version stable et les placer dans `/opt/pgsql/src`.

En tant qu'utilisateur **srcpostgres**, créer un répertoire dédié aux sources :

```
mkdir ~srcpostgres/src
cd ~/src
```

Aller sur [https://postgresql.org][https://www.postgresql.org/ftp/source/], cliquer *Download* et récupérer le lien vers l'archive des fichiers sources de la dernière version stable (PostgreSQL 14.3 au moment où ceci est écrit). Il est possible de le faire en ligne de commande :

```
wget https://ftp.postgresql.org/pub/source/v14.3/postgresql-14.3.tar.bz2
```

Il faut décompresser l'archive :

```
tar xjvf postgresql-14.3.tar.bz2
cd postgresql-14.3
```

Compilation et installation

> L'installation des binaires compilés se fera dans `/opt/pgsql/14/`.
> Configurer en conséquence l'environnement de compilation (`./configure`).
> Compiler PostgreSQL.

Configuration :

```
$ ./configure --prefix /opt/pgsql/14

checking build system type... x86_64-pc-linux-gnu
checking host system type... x86_64-pc-linux-gnu
checking which template to use... linux
checking whether NLS is wanted... no
checking for default port number... 5432
...
configure: creating ./config.status
config.status: creating GNUmakefile
config.status: creating src/Makefile.global
...
```

Des fichiers sont générés, notamment le *Makefile*.

La compilation se lance de manière classique. Elle peut prendre un certain temps sur les machines un peu anciennes :

```
$ make

make -C ./src/backend generated-headers
make[1] : on entre dans le répertoire « /opt/pgsql/postgresql-14.3/src/backend »
make -C catalog distprep generated-header-symlinks
make[2] : on entre dans le répertoire
                                « /opt/pgsql/postgresql-14.3/src/backend/catalog »
make[2]: Rien à faire pour « distprep ».
prereqdir=`cd './' >/dev/null && pwd` && \
cd '../../../src/include/catalog/' &&
for file in pg_proc_d.h pg_type_d.h pg_attribute_d.h ...
....
make[2] : on quitte le répertoire « /opt/pgsql/postgresql-14.3/src/test/perl »
make[1] : on quitte le répertoire « /opt/pgsql/postgresql-14.3/src »
make -C config all
make[1] : on entre dans le répertoire « /opt/pgsql/postgresql-14.3/config »
make[1]: Rien à faire pour « all ».
make[1] : on quitte le répertoire « /opt/pgsql/postgresql-14.3/config »
```

> Installer les fichiers obtenus.

L'installation peut se faire en tant que **srcpostgres** car nous avons défini comme cible le répertoire `/opt/pgsql/14/` qui lui appartient :

```
$ make install
```

Dans ce TP, nous nous sommes attachés à changer le moins possible d'utilisateur système. Il se peut que vous ayez à installer les fichiers obtenus en tant qu'utilisateur **root** dans d'autres environnements en fonction de la politique de sécurité adoptée.

> Où se trouvent les binaires installés de PostgreSQL ?

Les binaires installés sont situés dans le répertoire `/opt/pgsql/14/bin`.

Configurer le système

> Ajouter les variables d'environnement `PATH` et `LD_LIBRARY_PATH`

> au `~srcpostgres/.bash_profile` de l'utilisateur **srcpostgres** pour accéder facilement à ces binaires.

Ajouter les lignes suivantes à la fin du fichier `~srcpostgres/.bash_profile` (le nom de ce fichier peut être différent selon l'environnement utilisé) :

```
export PGDATA=/opt/pgsql/14/data
export PATH=/opt/pgsql/14/bin:$PATH
export LD_LIBRARY_PATH=/opt/pgsql/14/lib:$LD_LIBRARY_PATH
```

Il faut ensuite recharger le fichier à l'aide de la commande suivante (ne pas oublier le point et l'espace au début de la commande) :

```
. ~srcpostgres/.bash_profile
```

Vérifier que les chemins sont bons :

```
$ which psql
~/14/bin/psql
```

Création d'une instance

> Avec `initdb`, initialiser une instance dans `/opt/pgsql/14/data` en spécifiant **postgres** comme nom de super-utilisateur, et en activant les sommes de contrôle.

```
$ initdb -D $PGDATA -U postgres --data-checksums

The files belonging to this database system will be owned by user "srcpostgres".
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.

creating directory /opt/pgsql/14/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 ... Europe/Paris
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 /opt/pgsql/14/data -l logfile start
```

> Démarrer l'instance.

```
$ pg_ctl -D $PGDATA -l $PGDATA/server.log start

waiting for server to start.... done
server started

$ cat $PGDATA/server.log

LOG:  starting PostgreSQL 14.3 on x86_64-pc-linux-gnu,
      compiled by gcc (GCC) 8.5.0 20210514 (Red Hat 8.5.0-10), 64-bit
LOG:  listening on IPv6 address "::1", port 5432
LOG:  listening on IPv4 address "127.0.0.1", port 5432
LOG:  listening on Unix socket "/tmp/.s.PGSQL.5432"
LOG:  database system was shut down at 2022-05-17 11:53:32 UTC
LOG:  database system is ready to accept connections
```

> Tenter une première connexion avec `psql`.
> Pourquoi cela échoue-t-il ?

```
$ psql

psql: error: connection to server on socket "/tmp/.s.PGSQL.5432" failed:
 FATAL: role "srcpostgres" does not exist
```

Par défaut, `psql` demande à se connecter avec un nom d'utilisateur identique à celui en cours, mais la base de données ne connaît pas l'utilisateur **srcpostgres**. Par défaut, elle ne connaît que **postgres**.

> Se connecter en tant qu'utilisateur **postgres**. Ressortir.

```
$ psql -U postgres

psql (14.3)
Type "help" for help.
```

```
postgres=# exit
```

Noter que la connexion fonctionne parce que `pg_hba.conf` est par défaut très laxiste (méthode `trust` en local et *via* `localhost` !).

> Dans `.bash_profile`, configurer la variable d'environnement `PGUSER`.

Ajouter ceci à à la fin du fichier `~srcpostgres/.bash_profile` :

```
export PGUSER=postgres
```

Il faut ensuite recharger le fichier profile à l'aide de la commande suivante (ne pas oublier le point et l'espace au début de la commande) :

```
. ~srcpostgres/.bash_profile
```

Première base

> Créer une première base de donnée nommée `test`.

En ligne de commande shell :

```
$ createdb test
```

Alternativement, depuis `psql` :

```
postgres=# CREATE DATABASE test ;
```

```
CREATE DATABASE
```

> Se connecter à la base `test` et créer quelques tables.

```
psql test
```

```
test=# CREATE TABLE premieretable (x int) ;
CREATE TABLE
```

Arrêt

> Arrêter cette instance.

```
$ pg_ctl stop
```

```
waiting for server to shut down.... done
```

```
server stopped

$ tail $PGDATA/server.log

LOG:  received fast shutdown request
LOG:  aborting any active transactions
LOG:  background worker "logical replication launcher" exited with exit code 1
LOG:  shutting down
LOG:  database system is shut down
```

1.10.2 INSTALLATION DEPUIS LES PAQUETS BINAIRES DU PGDG

Pré-installation

> Quelle commande permet d'installer les paquets binaires de PostgreSQL ?

Tout dépend de votre distribution. Les systèmes les plus représentés sont Debian et ses dérivés (notamment Ubuntu), ainsi que Red Hat et dérivés (CentOS, Rocky Linux).

Le présent TP utilise Rocky Linux 8, basé sur une version communautaire qui se veut être le successeur du projet CentOS, interrompu en 2021[10] . Une version plus complète, ainsi que l'utilisation de paquets Debian, sont traités dans l'annexe « Installation de PostgreSQL depuis les paquets communautaires ».

> Quelle version est packagée ?

La dernière version stable de PostgreSQL disponible au moment de la rédaction de ce module est la version 14.3. Par contre, la dernière version disponible dans les dépôts dépend de votre distribution. C'est la raison pour laquelle **les dépôts du PGDG sont à privilégier**.

> Quels paquets devront également être installés ?

Le paquet `libpq` devra également être installé. À partir de la version 11, il est aussi nécessaire d'installer les paquets `llvmjit` (pour la compilation à la volée), qui réclame elle-même la présence du dépôt EPEL, mais c'est une fonctionnalité optionnelle qui ne sera pas traitée ici.

Installation

[10] https://blog.centos.org/2020/12/future-is-centos-stream

> Installer le dépôt.

Les commandes qui suivent sont testées sur Rocky Linux 8 et peuvent être consultées sur le guide de téléchargement PostgreSQL pour la famille Red Hat[11] .

Se connecter avec l'utilisateur `root` sur la machine virtuelle et recopier le script proposé par le guide. Dans la commande ci-dessous, les deux lignes **doivent être copiées et collées ensemble**.

```
dnf install -y https://download.postgresql.org\
/pub/repos/yum/reporpms/EL-8-x86_64/pgdg-redhat-repo-latest.noarch.rpm
```

> Désactiver le module d'installation pour la version PostgreSQL de la distribution.

Cette opération est inutile sous CentOS 7 mais nécessaire pour Rocky Linux 8.

```
dnf -qy module disable postgresql
```

> Installer les paquets de PostgreSQL14 : serveur, client, contribs.

```
dnf install -y postgresql14 postgresql14-server postgresql14-contrib
```

Il s'agit respectivement des outils clients, des binaires du serveur, des « contribs » et extensions (optionnelles mais chaudement conseillées). On met volontairement de côté le paquet `llvmjit`.

> Quel est le chemin des binaires ?

Ils se trouvent dans `/usr/pgsql-14/bin/` (chemin propre à ce packaging) :

```
$ ls -l /usr/pgsql-14/bin/

total 11092
-rwxr-xr-x. 1 root root   77576 Feb  9 08:14 clusterdb
-rwxr-xr-x. 1 root root   81712 Feb  9 08:14 createdb
-rwxr-xr-x. 1 root root   77944 Feb  9 08:14 createuser
...
-rwxr-xr-x. 1 root root   72672 Feb  9 08:14 pg_ctl
-rwxr-xr-x. 1 root root  427904 Feb  9 08:14 pg_dump
-rwxr-xr-x. 1 root root  115792 Feb  9 08:14 pg_dumpall
```

[11] https://www.postgresql.org/download/linux/redhat

```
...
-rwxr-xr-x. 1 root root 8403912 Feb  9 08:14 postgres
-rwxr-xr-x. 1 root root    2167 Feb  9 08:13 postgresql-14-check-db-dir
-rwxr-xr-x. 1 root root    9655 Feb  9 08:13 postgresql-14-setup
lrwxrwxrwx. 1 root root       8 Feb  9 08:13 postmaster -> postgres
-rwxr-xr-x. 1 root root  699624 Feb  9 08:14 psql
...
```

Noter qu'il existe des liens dans `/usr/bin` pointant vers la version la plus récente des outils en cas d'installation de plusieurs versions :

```
$ which psql
/usr/bin/psql

$ file /usr/bin/psql
/usr/bin/psql: symbolic link to /etc/alternatives/pgsql-psql

$ file /etc/alternatives/pgsql-psql
/etc/alternatives/pgsql-psql: symbolic link to /usr/pgsql-14/bin/psql
```

Création de la première instance

> Créer une première instance avec les outils de la famille Red Hat en activant les sommes de contrôle (*checksums*).

La création d'une instance passe par un outil spécifique à ces paquets. Il doit être appelé en tant que **root** (et non **postgres**). Optionnellement, on peut ajouter des paramètres d'initialisation à cette étape. La mise en place des sommes de contrôle est généralement conseillée pour être averti de toute corruption des fichiers.

Toujours en temps que **root** :

```
export PGSETUP_INITDB_OPTIONS="--data-checksums"
/usr/pgsql-14/bin/postgresql-14-setup initdb

Initializing database ... OK
```

> Vérifier ce qui a été fait dans le journal `initdb.log`.

La sortie de la commande précédente est redirigée vers le fichier `initdb.log` situé dans le répertoire qui contient celui de la base (`PGDATA`). Il est possible d'y vérifier l'ensemble des étapes réalisées, notamment l'activation des sommes de contrôle.

```
$ cat /var/lib/pgsql/14/initdb.log
```

```
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.

fixing permissions on existing directory /var/lib/pgsql/14/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

Success. You can now start the database server using:

    /usr/pgsql-14/bin/pg_ctl -D /var/lib/pgsql/14/data/ -l logfile start
```

Ne pas tenir compte de la dernière ligne, qui est une suggestion qui ne tient pas compte des outils prévus pour cet OS.

Démarrage

> Démarrer l'instance.

Attention, si vous avez créé une instance à partir des sources dans le TP précédent, elle doit impérativement être arrêtée pour pouvoir démarrer la nouvelle instance !
En effet, comme nous n'avons pas modifié le port par défaut (5432), les deux instances ne peuvent pas être démarrées en même temps, sauf à modifier le port dans la configuration de l'une d'entre elles.

En tant que **root** :

```
# systemctl start postgresql-14
```

Si aucune erreur ne s'affiche, tout va bien à priori.

Pour connaître l'état de l'instance :

```
# systemctl status postgresql-14
```

```
postgresql-14.service - PostgreSQL 14 database server
   Loaded: loaded (/usr/lib/systemd/system/postgresql-14.service; disabled;
           vendor preset: disabled)
   Active: active (running)
     Docs: https://www.postgresql.org/docs/14/static/
  Process: 54573 ExecStartPre=/usr/pgsql-14/bin/postgresql-14-check-db-dir
           ${PGDATA} (code=exited, status=0/SUCCESS)
 Main PID: 54580 (postmaster)
    Tasks: 8 (limit: 2749)
   Memory: 19.2M
   CGroup: /system.slice/postgresql-14.service
           ├─54580 /usr/pgsql-14/bin/postmaster -D /var/lib/pgsql/14/data/
           ├─54581 postgres: logger
           ├─54583 postgres: checkpointer
           ├─54584 postgres: background writer
           ├─54585 postgres: walwriter
           ├─54586 postgres: autovacuum launcher
           ├─54587 postgres: stats collector
           └─54588 postgres: logical replication launcher
```

> Activer le démarrage de l'instance au démarrage de la machine.

Le packaging Red Hat ne prévoie pas l'activation du service au boot, il faut le demander explicitement :

```
 # systemctl enable postgresql-14

Created symlink
  /etc/systemd/system/multi-user.target.wants/postgresql-14.service
  → /usr/lib/systemd/system/postgresql-14.service.
```

> Où sont les fichiers de données (`PGDATA`), et les traces de l'instance ?

Les données et fichiers de configuration sont dans `/var/lib/pgsql/14/data/`.

```
ls -l /var/lib/pgsql/14/data/

total 68
drwx------. 5 postgres postgres    41 Mar  2 10:35 base
...
drwx------. 2 postgres postgres    32 Mar  2 10:37 log
...
drwx------. 3 postgres postgres    60 Mar  2 10:35 pg_wal
...
```

```
-rw-------. 1 postgres postgres    88 Mar  2 10:35 postgresql.auto.conf
-rw-------. 1 postgres postgres 28750 Mar  2 10:35 postgresql.conf
...
```

Les traces sont dans le sous-répertoire `log`.

```
ls -l /var/lib/pgsql/14/data/log/

total 4
-rw-------. 1 postgres postgres 709 Mar  2 10:37 postgresql-Wed.log
```

Configuration

> Vérifier la configuration par défaut de PostgreSQL. Est-ce que le serveur écoute sur le réseau ?

Il est possible de vérifier dans le fichier `postgresql.conf` que par défaut, le serveur écoute uniquement l'interface réseau `localhost` (la valeur est commentée mais c'est bien celle par défaut) :

```
$ grep listen_addresses /var/lib/pgsql/14/data/postgresql.conf

#listen_addresses = 'localhost'         # what IP address(es) to listen on;
```

Il faudra donc modifier cela pour que des utilisateurs puissent se connecter depuis d'autres machines.

Il est aussi possible de vérifier au niveau système en utilisant la commande `netstat` (qui nécessite l'installation du paquet `net-tools`) :

```
$ netstat -anp | grep postmaster

tcp     0  0 127.0.0.1:5432   0.0.0.0:*         LISTEN      28028/postmaster
tcp6    0  0 ::1:5432         :::*              LISTEN      28028/postmaster
udp6    0  0 ::1:57123        ::1:57123         ESTABLISHED 28028/postmaster
unix 2 [ ACC ] STREAM LISTENING 301922 28028/postmaster /tmp/.s.PGSQL.5432
unix 2 [ ACC ] STREAM LISTENING 301920 28028/postmaster
                                          /var/run/postgresql/.s.PGSQL.5432
```

> Quel est l'utilisateur sous lequel tourne l'instance ?

C'est l'utilisateur nommé **postgres** :

```
$ ps -U postgres -f -o pid,user,cmd

    PID USER     CMD
```

```
  38465 postgres /usr/pgsql-14/bin/postmaster -D /var/lib/pgsql/14/data/
  38468 postgres  \_ postgres: logger
  38470 postgres  \_ postgres: checkpointer
  38471 postgres  \_ postgres: background writer
  38472 postgres  \_ postgres: walwriter
  38473 postgres  \_ postgres: autovacuum launcher
  38474 postgres  \_ postgres: stats collector
  38475 postgres  \_ postgres: logical replication launcher
```

Il possède aussi le `PGDATA` :

```
$ ls -l /var/lib/pgsql/14/

total 8
drwx------.  2 postgres postgres    6 Feb  9 08:13 backups
drwx------. 20 postgres postgres 4096 Mar  4 16:18 data
-rw-------.  1 postgres postgres  910 Mar  2 10:35 initdb.log
```

postgres est le nom traditionnel sur la plupart des distributions, mais il n'est pas obligatoire (par exemple, le TP par compilation utilise un autre utilisateur).

Connexion

> En tant que **root**, tenter une connexion avec `psql`.

```
 # psql

psql: FATAL:  role "root" does not exist
```

Cela échoue car `psql` tente de se connecter avec l'utilisateur système en cours, soit **root**. Ça ne marchera pas mieux cependant en essayant de se connecter avec l'utilisateur **postgres** :

```
 # psql -U postgres

psql: FATAL:  Peer authentication failed for user "postgres"
```

En effet, le `pg_hba.conf` est configuré de telle manière que l'utilisateur de PostgreSQL et celui du système doivent porter le même nom.

> En tant que **postgres**, tenter une connexion avec `psql`.
> Quitter.

```
sudo -iu postgres psql

psql (14.3)
Type "help" for help.
```

```
postgres=# exit
```

La connexion fonctionne aussi depuis l'utilisateur **dalibo** puisqu'il peut effectuer un `sudo`.

> À quelle base se connecte-t-on par défaut ?

```
sudo -iu postgres psql
```

```
psql (14.3)
Type "help" for help.

postgres=# \conninfo
  You are connected to database "postgres" as user "postgres"
  via socket in "/var/run/postgresql" at port "5432".
```

Là encore, la présence d'une base nommée `postgres` est une tradition et non une obligation.

Première base

> Créer une première base de données et y créer des tables.

```
$ sudo -iu postgres psql
```

```
postgres=# CREATE DATABASE test ;
CREATE DATABASE
```

Alternativement :

```
$ sudo -iu postgres createdb test
```

Se connecter explicitement à la bonne base :

```
$ sudo -iu postgres psql -d test
```

```
test=# CREATE TABLE mapremieretable (x int);
CREATE TABLE

test=# \d+
                            Liste des relations
 Schéma |       Nom       | Type  | Propriétaire | Taille  | Description
--------+-----------------+-------+--------------+---------+-------------
 public | mapremieretable | table | postgres     | 0 bytes |
```

1.11 INSTALLATION DE POSTGRESQL DEPUIS LES PAQUETS COMMUNAUTAIRES

L'installation est détaillée ici pour Rocky Linux 8 (similaire à Red Hat 8), Red Hat/CentOS 7, et Debian/Ubuntu.

Elle ne dure que quelques minutes.

1.11.1 SUR ROCKY LINUX 8

Installation du dépôt communautaire :

Sauf précision, tout est à effectuer en tant qu'utilisateur **root**.

Les dépôts de la communauté sont sur https://yum.postgresql.org/. Les commandes qui suivent peuvent être générées par l'assistant sur https://www.postgresql.org/download/linux/redhat/, en précisant :

- la version majeure de PostgreSQL (ici la 14) ;
- la distribution (ici Rocky Linux 8) ;
- l'architecture (ici x86_64, la plus courante).

Il faut installer le dépôt et désactiver le module PostgreSQL par défaut :

```
# dnf install -y https://download.postgresql.org/pub/repos/yum/reporpms\
/EL-8-x86_64/pgdg-redhat-repo-latest.noarch.rpm

# dnf -qy module disable postgresql
```

Installation de PostgreSQL 14 :

```
# dnf install -y postgresql14-server postgresql14-contrib
```

Les outils clients et les librairies nécessaires seront automatiquement installés.

Tout à fait optionnellement, une fonctionnalité avancée, le JIT (*Just In Time compilation*), nécessite un paquet séparé, qui lui-même nécessite des paquets du dépôt EPEL :

```
# dnf install postgresql14-llvmjit
```

Création d'une première instance :

Il est conseillé de déclarer `PG_SETUP_INITDB_OPTIONS`, notamment pour mettre en place les sommes de contrôle et forcer les traces en anglais :

```
# export PGSETUP_INITDB_OPTIONS='--data-checksums --lc-messages=C'
# /usr/pgsql-14/bin/postgresql-14-setup initdb
# cat /var/lib/pgsql/14/initdb.log
```

Ce dernier fichier permet de vérifier que tout s'est bien passé.

Chemins :

Objet	Chemin
Binaires	`/usr/pgsql-14/bin`
Répertoire de l'utilisateur **postgres**	`/var/lib/pgsql`
`PGDATA` par défaut	`/var/lib/pgsql/14/data`
Fichiers de configuration	dans `PGDATA/`
Traces	dans `PGDATA/log`

Configuration :

Modifier `postgresql.conf` est facultatif pour un premier essai.

Démarrage/arrêt de l'instance, rechargement de configuration :

```
# systemctl start postgresql-14
# systemctl stop postgresql-14
# systemctl reload postgresql-14
```

Test rapide de bon fonctionnement

```
# systemctl --all |grep postgres
# sudo -iu postgres psql
```

Démarrage de l'instance au démarrage du système d'exploitation :

```
# systemctl enable postgresql-14
```

Consultation de l'état de l'instance :

```
# systemctl status postgresql-14
```

Ouverture du *firewall* pour le port 5432 :

Si le *firewall* est actif (dans le doute, consulter `systemctl status firewalld`) :

```
# firewall-cmd --zone=public --add-port=5432/tcp --permanent
# firewall-cmd --reload
# firewall-cmd --list-all
```

Création d'autres instances :

Si des instances de *versions majeures différentes* doivent être installées, il faudra installer les binaires pour chacune, et l'instance par défaut de chaque version vivra dans un sous-répertoire différent de `/var/lib/pgsql` automatiquement créé à l'installation. Il faudra juste modifier les ports dans les `postgresql.conf`.

Si plusieurs instances d'une même version majeure (forcément de la même version mineure) doivent cohabiter sur le même serveur, il faudra les installer dans des `PGDATA` différents.

- Ne pas utiliser de tiret dans le nom d'une instance (problèmes potentiels avec systemd).
- Respecter les normes et conventions de l'OS : placer les instances dans un sous-répertoire de `/var/lib/pgsqsl/14/` (ou l'équivalent pour d'autres versions majeures).
- Création du fichier service de la deuxième instance :

```
# cp /lib/systemd/system/postgresql-14.service \
       /etc/systemd/system/postgresql-14-secondaire.service
```

- Modification du fichier avec le nouveau chemin :

```
Environment=PGDATA=/var/lib/pgsql/14/secondaire
```

- Option 1 : création d'une nouvelle instance vierge :

```
# /usr/pgsql-14/bin/postgresql-14-setup initdb postgresql-14-secondaire
```

- Option 2 : restauration d'une sauvegarde : la procédure dépend de votre outil.
- Adaptation de `postgresql.conf` (port !), `recovery.conf`...
- Commandes de maintenance :

```
# systemctl [start|stop|reload|status] postgresql-14-secondaire
# systemctl [enable|disable] postgresql-14-secondaire
```

- Ouvrir un port dans le firewall au besoin.

1.11.2 SUR RED HAT 7 / CENT OS 7

Fondamentalement, le principe reste le même qu'en version 8. Il faudra utiliser `yum` plutôt que `dnf`.

ATTENTION : Red Hat et CentOS 6 et 7 fournissent par défaut des versions de PostgreSQL qui ne sont plus supportées. Ne jamais installer les packages `postgresql`, `postgresql-client` et `postgresql-server` !
L'utilisation des dépôts du PGDG est donc obligatoire.

Il n'y a pas besoin de désactiver de module AppStream.

Le JIT (*Just In Time compilation*), nécessite un paquet séparé, qui lui-même nécessite des paquets du dépôt EPEL :

```
# yum install epel-release
# yum install postgresql14-llvmjit
```

La création de l'instance et la suite sont identiques.

1.11.3 SUR DEBIAN / UBUNTU

Sauf précision, tout est à effectuer en tant qu'utilisateur **root**.

Installation du dépôt communautaire :

Référence : https://apt.postgresql.org/

- Import des certificats et de la clé :

```
# apt install curl ca-certificates gnupg
# curl https://www.postgresql.org/media/keys/ACCC4CF8.asc | apt-key add -
```

- Création du fichier du dépôt `/etc/apt/sources.list.d/pgdg.list` (ici pour Debian 11 « bullseye » ; adapter au nom de code de la version de Debian ou Ubuntu correspondante : **stretch**, **bionic**, **focal**...) :

```
deb http://apt.postgresql.org/pub/repos/apt bullseye-pgdg main
```

Installation de PostgreSQL 14 :

La méthode la plus propre consiste à modifier la configuration par défaut avant l'installation :

```
# apt update
# apt install postgresql-common
```

Dans `/etc/postgresql-common/createcluster.conf`, paramétrer au moins les sommes de contrôle et les traces en anglais :

```
initdb_options = '--data-checksums --lc-messages=C'
```

Puis installer les paquets serveur et clients et leurs dépendances :

```
# apt install postgresql-14 postgresql-client-14
```

(Pour les versions 9.x, installer aussi le paquet `postgresql-contrib-9.x`).

La première instance est automatiquement créée, démarrée et déclarée comme service à lancer au démarrage du système. Elle est immédiatement accessible par l'utilisateur système **postgres**.

Chemins :

Objet	Chemin

Binaires	`/usr/lib/postgresql/14/bin/`
Répertoire de l'utilisateur **postgres**	`/var/lib/postgresql`
PGDATA de l'instance par défaut	`/var/lib/postgresql/14/main`
Fichiers de configuration	dans `/etc/postgresql/14/main/`
Traces	dans `/var/log/postgresql/`

Configuration

Modifier `postgresql.conf` est facultatif pour un premier essai.

Démarrage/arrêt de l'instance, rechargement de configuration :

Debian fournit ses propres outils :

```
# pg_ctlcluster 14 main [start|stop|reload|status]
```

Démarrage de l'instance au lancement :

C'est en place par défaut, et modifiable dans `/etc/postgresql/14/main/start.conf`.

Ouverture du firewall :

Debian et Ubuntu n'installent pas de firewall par défaut.

Statut des instances :

```
# pg_lsclusters
```

Test rapide de bon fonctionnement

```
# systemctl --all |grep postgres
# sudo -iu postgres psql
```

Destruction d'une instance :

```
# pg_dropcluster 14 main
```

Création d'autres instances :

Ce qui suit est valable pour remplacer l'instance par défaut par une autre, par exemple pour mettre les *checksums* en place :

- les paramètres de création d'instance dans `/etc/postgresql-common/createcluster.conf` peuvent être modifiés, par exemple ici pour : les *checksums*, les messages en anglais, l'authentification sécurisée, le format des traces et un emplacement séparé pour les journaux :

```
initdb_options = '--data-checksums --lc-messages=C --auth-host=scram-sha-256 --auth-local=peer'
log_line_prefix = '%t [%p]: [%l-1] user=%u,db=%d,app=%a,client=%h '
waldir = '/var/lib/postgresql/wal/%v/%c/pg_wal'
```

- création de l'instance, avec possibilité là aussi de préciser certains paramètres du `postgresql.conf` voire de modifier les chemins des fichiers (déconseillé si vous pouvez l'éviter) :

```
# pg_createcluster 14 secondaire \
  --port=5433 \
  --datadir=/PGDATA/11/basedecisionnelle \
  --pgoption shared_buffers='8GB' --pgoption work_mem='50MB' \
  --  --data-checksums --waldir=/ssd/postgresql/11/basedecisionnelle/journaux
```

- démarrage :

```
# pg_ctlcluster 14 secondaire start
```

1.11.4 ACCÈS À L'INSTANCE

Par défaut, l'instance n'est accessible que par l'utilisateur système `postgres`, qui n'a pas de mot de passe. Un détour par `sudo` est nécessaire :

```
$ sudo -iu postgres psql
psql (14.0)
Saisissez « help » pour l'aide.
postgres=#
```

Ce qui suit permet la connexion directement depuis un utilisateur du système :

Pour des tests (pas en production !), il suffit de passer à `trust` le type de la connexion en local dans le `pg_hba.conf` :

```
local   all             postgres                                trust
```

La connexion en tant qu'utilisateur `postgres` (ou tout autre) n'est alors plus sécurisée :

```
dalibo:~$ psql -U postgres
psql (14.0)
Saisissez « help » pour l'aide.
postgres=#
```

Une authentification par mot de passe est plus sécurisée :

- dans `pg_hba.conf`, mise en place d'une authentification par mot de passe (`md5` par défaut) pour les accès à `localhost` :

```
# IPv4 local connections:
host    all             all             127.0.0.1/32            md5
```

```
# IPv6 local connections:
host    all             all             ::1/128                 md5
```

(une authentification `scram-sha-256` est plus conseillée mais elle impose que `password_encryption` soit à cette valeur dans `postgresql.conf` avant de définir les mots de passe).

- ajout d'un mot de passe à l'utilisateur `postgres` de l'instance ;

```
dalibo:~$ sudo -iu postgres psql
psql (14.0)
Saisissez « help » pour l'aide.
postgres=# \password
Saisissez le nouveau mot de passe :
Saisissez-le à nouveau :
postgres=# \q

dalibo:~$ psql -h localhost -U postgres
Mot de passe pour l'utilisateur postgres :
psql (14.0)
Saisissez « help » pour l'aide.
postgres=#
```

- pour se connecter sans taper le mot de passe, un fichier `.pgpass` dans le répertoire personnel doit contenir les informations sur cette connexion :

```
localhost:5432:*:postgres:motdepassetrèslong
```

- ce fichier doit être protégé des autres utilisateurs :

```
$ chmod 600 ~/.pgpass
```

- pour n'avoir à taper que `psql`, on peut définir ces variables d'environnement dans la session voire dans `~/.bashrc` :

```
export PGUSER=postgres
export PGDATABASE=postgres
export PGHOST=localhost
```

Rappels :

- en cas de problème, consulter les traces (dans `/var/lib/pgsql/14/data/log` ou `/var/log/postgresql/`) ;
- toute modification de `pg_hba.conf` implique de recharger la configuration par une de ces trois méthodes selon le système :

```
root:~# systemctl reload postgresql-14

root:~# pg_ctlcluster 14 main reload
```

```
postgres:~$ psql -c 'SELECT pg_reload_conf();'
```

NOTES

NOTES

NOTES

NOS AUTRES PUBLICATIONS

FORMATIONS

- **DBA1 : Administration PostgreSQL**
  https://dali.bo/dba1
- **DBA2 : Administration PostgreSQL avancé**
  https://dali.bo/dba2
- **DBA3 : Sauvegarde et réplication avec PostgreSQL**
  https://dali.bo/dba3
- **DEVPG : Développer avec PostgreSQL**
  https://dali.bo/devpg
- **PERF1 : PostgreSQL Performances**
  https://dali.bo/perf1
- **PERF2 : Indexation et SQL avancés**
  https://dali.bo/perf2
- **MIGORPG : Migrer d'Oracle à PostgreSQL**
  https://dali.bo/migorpg
- **HAPAT : Haute disponibilité avec PostgreSQL**
  https://dali.bo/hapat

LIVRES BLANCS

- **Migrer d'Oracle à PostgreSQL**
- **Industrialiser PostgreSQL**
- **Bonnes pratiques de modélisation avec PostgreSQL**
- **Bonnes pratiques de développement avec PostgreSQL**

TÉLÉCHARGEMENT GRATUIT

Les versions électroniques de nos publications sont disponibles gratuitement sous licence open-source ou sous licence Creative Commons. Contactez-nous à l'adresse contact@dalibo.com pour plus d'information.

DALIBO, L'EXPERTISE POSTGRESQL

Depuis 2005, DALIBO met à la disposition de ses clients son savoir-faire dans le domaine des bases de données et propose des services de conseil, de formation et de support aux entreprises et aux institutionnels.

En parallèle de son activité commerciale, DALIBO contribue aux développements de la communauté PostgreSQL et participe activement à l'animation de la communauté francophone de PostgreSQL. La société est également à l'origine de nombreux outils libres de supervision, de migration, de sauvegarde et d'optimisation.

Le succès de PostgreSQL démontre que la transparence, l'ouverture et l'auto-gestion sont à la fois une source d'innovation et un gage de pérennité. DALIBO a intégré ces principes dans son ADN en optant pour le statut de SCOP : la société est contrôlée à 100 % par ses salariés, les décisions sont prises collectivement et les bénéfices sont partagés à parts égales.