# PostgreSQL

## Généralités

PostgreSQL est un serveur de base de données (SGBDR) libre (licence BSD) et qui est un des meilleurs système de base de données au monde. Il dépasse MySQL sur de nombreux points mais est moins présent que celui-ci en entreprise.

## Documentations

- Site officiel : http://www.postgresql.org/
- Site communauté française : http://www.postgresql.fr/

## Installation

```
yum install postgresql-server
yum install postgresql

aptitude install postgresql
```

## Configuration

- La configuration se situe entièrement dans le fichier `postgresql.conf`.
- La gestion des droits d'accès à la base de données se situe dans le fichier `pg_hba.conf`.

```
cd /etc/postgresql/9.1/main
cp -p postgresql.conf postgresql.conf.orig
```

### postgresql.conf

`/etc/postgresql/9.1/main/postgresql.conf`

Toutes les options du fichier de configuration sont surchargeables par les options passées à l'exécutable en ligne de commande.

- Tuning configuration : https://wiki.postgresql.org/wiki/Tuning_Your_PostgreSQL_Server
- `listen_addresses = '*'` = écoute sur toutes les interfaces réseaux de la machines. On peut spécifier des adresses réseaux. Exemple : `listen_addresses = '<@IP1> <@IP2>'`
- `shared_buffers` = mémoire max qui pourra être utilisé par postgresql. Si il en a pas besoin, il n'utilisera pas tout. Ce n'est pas comme Oracle si on lui dit 1Go, il prend 1Go.

Pour un serveur dédié à la base de données, il est conseillé de prendre 50% de la RAM du système. La plupart du temps 400Mo à 1Go.

Par défaut il est configuré à 32Mo de RAM. Il est très important de positionner cette valeur.

- work_mem : mémoire à utiliser pour les opérations de tri interne et pour les tables de découpage avant de basculer sur des fichiers temporaires sur disque.

10Mega est une bonne valeur de départ. Par défaut c'est 1Mega.

Faire attention entre le nombre de connexions possibles et le work_mem car pour 100 connexions simultanées c'est 100*work_mem donc 100*10 = 1Go de RAM.

2000 connexions concurrentes c'est le maxi. Pour plus il faut mettre un pooler.

- maintenance_work_mem = jusqu'à 512Mega.

Très intéressant lorsqu'on fait des opérations de maintenance car ca permet de moins solliciter la mémoire dédié à postgreSQL pour ses traitements.

Valeurs appliquées.

```
#listen_addresses = 'localhost'
shared_buffers = 384MB
work_mem = 10MB
```

Il est nécessaire de monter la taille maximum de la mémoire partagée du kernel pour appliquer la modification du shared_buffers. Ajouter les lignes qui suivent dans le fichier /etc/sysctl.conf pour passer la valeur à 500 Mo.

```
kernel.shmmax=512000000
```

Pour appliquer cette valeur sans redémarrage.

```
sysctl -w kernel.shmmax=512000000
```

## pg_hba.conf

/etc/postgresql/9.1/main/pg_hba.conf

```
local   all             postgres                                peer
local   all             mypguser                                trust
```

## Utilisateur / base de données

Le compte système postgres est locké par défaut. IL ne faut pas attribuer de mot de passe à ce compte système postgres créé à l'installation. Par contre, il faut en créer un au compte postgres de PostgreSQL dans la base de données.

```
su - postgres
psql
\password postgres
Enter new password:
Enter it again:
```

L'idée de base peu importe les services, c'est de conserver l'admin de base de PostgreSQL et ne pas s'en servir. A côté, on créé un administrateur pour chaque base de données à créer. On doit commencer par se loguer avec l'admin PostgreSQL.

```
su - postgres
```

Ensuite, on créé un utilisateur de base.

```
createuser -P mypguser
Enter password for new role:
Enter it again:
Shall the new role be a superuser? (y/n) n
Shall the new role be allowed to create databases? (y/n) n
Shall the new role be allowed to create more new roles? (y/n) n
```

Il est aussi possible de réaliser la même opération en commandes SQL.

```
su - postgres
psql
postgres=# CREATE USER mypguser WITH PASSWORD 'mypguserpass';
postgres=# CREATE DATABASE mypgdatabase OWNER mypguser;
```

Création d'une BD dont l'utilisateur nouvellement créé est le owner (-O).

```
createdb -O mypguser mypgdatabase
```

Se connecter à la base avec notre utilisateur.

```
# su - mypguser
$ psql mypgdatabase
ou directement
psql -d mypgdatabase -U mypguser
```

## Backup

- http://www.gokhanatil.com/2014/12/how-to-backup-and-restore-postgresql-9-3-databases.html

La sauvegarde se fait soit en SQL pour l'extraction des données (dump), soit à chaud grâce à un snapshot qui s'appuie sur les WAL. C'est cette solution qui est décrite et qui permet donc de sauvegarder sans arrêter la base de données mais d'obtenir une sauvegarde consistante.

/etc/profile

```
export PGDATA=<data_directory_postgresqlconf_value>
```

Création d'un dossier qui contiendra les copies des archives logs.

```
mkdir /data/postgresql-wal
chown postgres:postgres /data/postgresql-wal
```

PostgreSQL supports Write Ahead Log (WAL) mechanism like Oracle. So everything will be written to (redo)logs before they written into actual datafiles. So we will use a similar method to Oracle. We need to start “the backup mode”, copy the (data) files, and stop the backup mode, and add the archived logs to our backup. There are SQL commands for starting backup mode (pg_start_backup) and for stopping backup mode (pg_stop_backup), and we can copy the files using OS commands. Good thing is, since 9.1, PostgreSQL comes with a backup tool named “pg_basebackup”. It’ll do everything for us.

To be able to use pg_basebackup, we need to turn on archiving and also turn on wal_sender process. Archiving controlled by “archive_mode”, “archive_command” and “wal_level” parameters. An interesting thing about PostgreSQL is, you need to write OS commands to copy redologs (WAL) files to an archive location. While writing a copy command, you can use %p and %f variables. %p variable holds the full path of wal file, and %f holds only the file name. So something like “cp %p /data/postgresql-wal/%f”, will copy the wal file to /archives directory.

WAL Sender process is used to feed logs to a replicate database, and pg_basebackup also uses it to receive archive logs. It’s controlled by max_wal_senders parameter. We’ll set it to 1, so our pg_basebackup will be able to connect PostgreSQL server to fetch the required WAL files (the ones created during backup period).

Activation des archives logs (WAL) dans postgresql.conf.

```
max_wal_senders=1
wal_level=hot_standby
archive_mode=on
archive_command='cp %p /data/postgresql-wal/%f'
```

Autoriser cette opération dans le fichier d'autorisation pg_hba.conf.

```
local    replication     postgres        trust
```

Commande de sauvegarde à chaud et consistante, disponible à partir de PostgreSQL 9.1.

```
pg_basebackup -U postgres --xlog --format=t -z -D <backup_path>/`date
+%Y%m%d-%Hh%M`
```

It will create a directory such as 20150308-20h00 (depending the date it’s executed), compress all database files into a TAR file (–format=t) with gzip compression (-z) and put it into that folder. The name of backup file will be base.tar. This TAR file will contain the required WAL files (–xlog) to be able to recover the database. It even contains postgresql.conf and pg_hba.conf.

You may write a script to compress the archived log files and keep them with your base backups. After backing up, you better delete the old archived files. Here’s a sample script to backup and clean the archive logs older than 2 days, do not forget the change the folder name according to your real

archive location:

```
tar -cvzf <backup_path>/wal.tar.gz /data/postgresql-wal/
find /data/postgresql-wal/* -mtime +2 -exec ls -l {} \;
```

Les archives logs nécessaires à la restauration exécutée au moment de la sauvegarde sont présent dans l'archive base.tar.gz. Les archives logs sauvegardées à part dans le wal.tar.gz servent à pouvoir restaurer après le moment de la sauvegarde. Ils permettent d'être plus fin dans la restauration.

### Restauration

All we need is to extract tar file into the $PGDATA folder. Make sure that PostgreSQL services are down before you copy the files. If we want to restore the databases on a new server, we need to install PostgreSQL software first. So we restored our database files, and have required wal (redolog) files to recover the database, but if we open the database now, we'll lose the data since the backup is done. We may want to apply the logs which are created after the backup. To be able to do it, create a "recovery.conf" file which has the below line:

```
cd /backups/20150308-20h00
tar -xf base.tar -C /postgres/
```

Contenu du fichier recovery.conf

```
restore_command = 'cp /data/postgresql-wal/%f %p'
```

When we issue "pg_ctl start", PostgreSQL will see the "recovery.conf" file and start recovering the database. When the recovery is done, the "recovery.conf" file will be renamed to "recovery.done".

## Impacts de la configuration sur la proportion relative à la charge de la base de données

50% de la charge.

- Le serveur qui héberge le SGBDR : le matériel, la distribution et le kernel.
- Le moteur de la base de données : postgresql.conf.
- La base de données : l'organisation des fichiers de PostgreSQL.

50% de la charge.

- L'application en elle-même : le schéma et les requêtes.

## Les quatres tâches de maintenance PostgreSQL

- vacuum
- analyse
- cluster

- index

## Opérations de maintenance

Aucune base de données PostgreSQL ne sera performante s'il n'y a pas d'exécutions régulières de :

- VACUUM
- VACUUM FULL
- ANALYZE : Plusieurs fois par jour, voire heures pour des tables à forte transactionalité.
- AUTOVACUUM

La fonctionnalité d'autovacuum permet de s'affranchir des trois précédentes. Toutefois, un usage régulier de vacuum full et analyze permet de confirmer la bonne configuration de l'autovacuum. Pensez à utiliser l'utilitaire en ligne de commande `vacuumdb` pour programmer tous ces travaux dans des crontabs. Comme pour tout utilitaire associé à PostgreSQL, lancez la commande suivante pour avoir une aide complète : `$ vacuumdb –help`

Vacuum = nettoyeur des relations postgresql. Repérer dans toutes les tables quels sont les espaces disponibles qui peuvent être récupérés.

FSM ( Free Space Map) : taille de la fsm partagée qui contient la localisation des espaces non utilisées dans la base de données. PostgreSQL ne fera qu'augmenter au niveau de la taille. Le vacuum full récupère de l'espace sur le disque. Il est absolument nécessaire si le FSM est mal taillé.

`vacuum_cost_delay` = à un certain cout de ressources, cette valeur va arrêter le vacuum. Ca évite de saturer le serveur.

## reindex

- Permet de provoquer une réindexation complète de la base. Utiliser pour cela la commande (SQL ou shell) sans arguments. On peut aussi demander la réindexation d'une table particulière.
- Permet aussi de regénérer les index des catalogues système. Cette fonctionnalité est d'ailleurs fort utile lors de la corruption d'un index (fréquent si la base est maltraitée). Il faut alors que la base soit utilisée en mode autonome.
- Impact minime sur la charge du serveur (load). L'index re-créé remplace l'index actuel une fois seulement qu'il est complètement généré. On a ainsi l'assurance qu'à tout moment les requêtes SQL entrantes peuvent utiliser tous les index, y compris ceux en cours de mise à jour.

## cluster

CLUSTER nom_index ON nom_table ;

- Permet d'appliquer la structure d'un index aux données d'une table, qui seront alors organisées physiquement comme lui. La clusterisation est un moyen facile d'obtenir un gain de performances rapide dans le cas où les données seront lues en fonction de l'organisation de l'index : tous les blocs de données sont contigus sur le disque.
- La commande est one-shot ! La commande est à initier régulièrement, lorsque la variation en pourcentage des lignes d'une base est au-delà d'un seuil… qu'il faut trouver.
- Cependant, la commande appelée sans arguments permet de reproduire l'opération sur toutes

les tables possédant un CLUSTER.
- Hélas, verrou ACCESS EXCLUSIVE… Un verrou ACCESS EXCLUSIVE est créé pour la table en cours de clusterisation : elle ne sera alors pas accessible, ni en lecture, ni en écriture et pour tous les utilisateurs, le temps de l'opération.

# Exemples de commandes

## Vacuumdb

Vacuumdb permet de faire un vacuum, un vacuum full et un analyze. Cette commande peut être positionné automatiquement dans `postgresql.conf` via `autovacuum`.

```
vacuumdb --help
```

Exemple.

```
vacuumdb --all  --full -h HOSTNAME -p 5432 -U user -W
vacuumdb --all  --analyze -h HOSTNAME -p 5432 -U user -W
```

## Reindex

Reindex permet de réindexer la base de données.

```
reindexdb --help
```

Exemple.

```
reindex --all --echo -h HOSTNAME -p 5432 -U user -W
```

## Cluster

Cluster permet d’appliquer la structure d’un index aux données d’une table, qui seront alors organisées physiquement comme lui.

```
clusterdb --help
```

Exemple.

```
clusterdb --all --echo -h HOSTNAME -p 5432 -U user -W
```

# Outils

- dbDesigner : outil de visualisation de schéma de base et d'export de schéma au format XML.