DALIBO

L'expertise PostgreSQL

# Installation de PostgreSQL

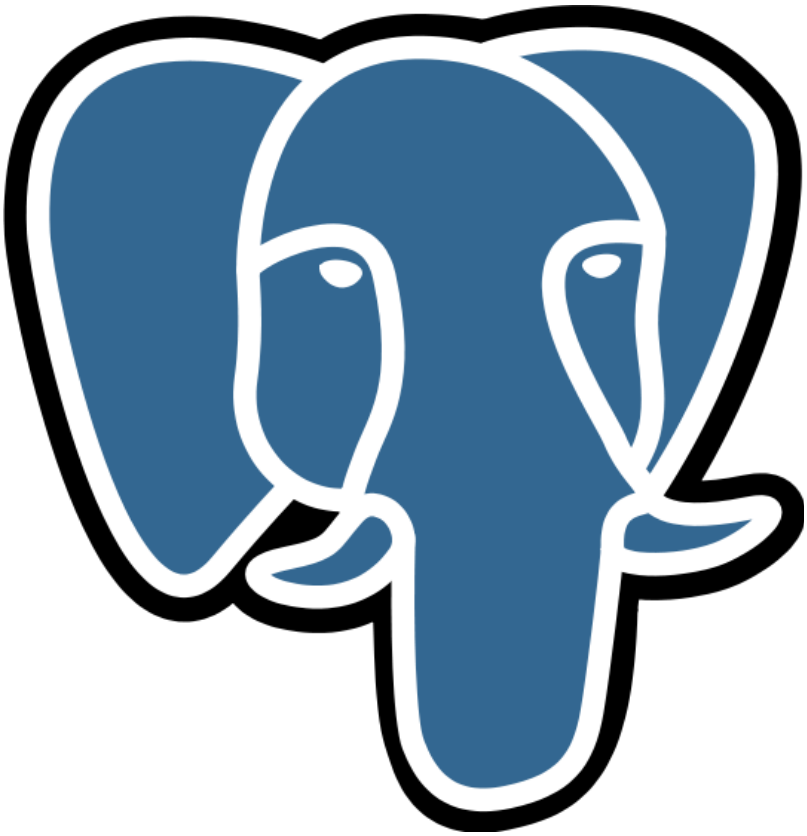


Pour toute information complémentaire, contactez :
formation@dalibo.com

# Table des matières


Installation de PostgreSQL....4
1 Introduction....5
1.1 Licence Creative Commons CC-BY-NC-SA....5
2 Installation à partir des sources....7
2.1 Téléchargement....7
2.2 Phases de compilation/installation....12
2.3 Options pour ./configure....14
2.4 Tests de régression....15
2.5 Création de l'utilisateur....16
2.6 Création du répertoire de données....17
2.7 Lancement et arrêt....19
3 Installation à partir des paquets Linux....21
3.1 Paquets Debian officiels....21
3.2 Paquets Debian communautaires....22
3.3 Paquets RedHat officiels....24
3.4 Paquets RedHat communautaires....25
4 Installation sous Windows....27
4.1 Installeur graphique - bienvenue....28
4.2 Installeur graphique - répertoire d'installation....29
4.3 Installeur graphique - répertoire données....30
4.4 Installeur graphique - mot de passe....31
4.5 Installeur graphique - port....32
4.6 Installeur graphique - autres....33
4.7 Installeur graphique - prêt....34
4.8 Installeur graphique - terminé....35
5 Premiers réglages....36
5.1 Sécurité....36
5.2 Configuration minimale....37
5.3 Configuration précédence des paramètres....38
5.4 Configuration des connexions....38
5.5 Configuration des ressources mémoire....39
5.6 Configuration des journaux de transactions 1/2....40
5.7 Configuration des journaux de transactions 2/2....40
5.8 Configuration des traces....41
5.9 Configuration des démons....42
6 Mise à jour....43
6.1 Recommandations....43
6.2 Mise à jour mineure....44
6.3 Mise à jour majeure....45
6.4 Mise à jour majeure par dump/restore....45
6.5 Mise à jour majeure par Slony....46
6.6 Mise à jour majeure par pg_upgrade....46
7 Conclusion....48
7.1 Pour aller plus loin....48
7.2 Questions....48

# Installation de PostgreSQL



# 1 Introduction

- Installation depuis les sources
- Installation depuis les binaires
  - installation à partir de packages
  - 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.

## 1.1 Licence Creative Commons CC-BY-NC-SA

# 2 Installation à partir des sources

Étapes :

- Téléchargement
- Vérification des pré-requis
- 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.

## 2.1 Téléchargement

- Disponible via :
  - HTTP
  - FTP
- Télécharger le fichier postgresql-X.Y.Z.tar.bz2

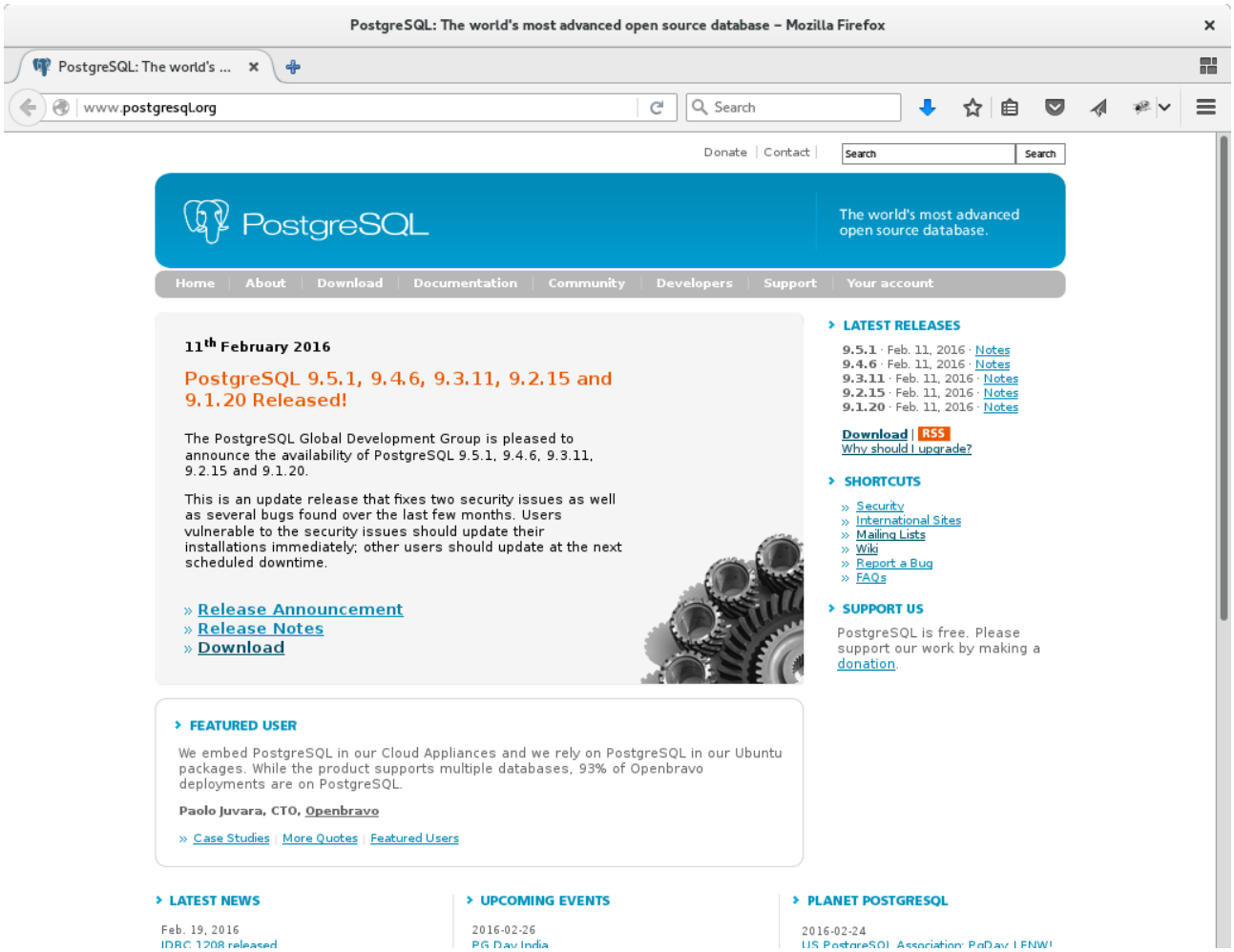
Les fichiers sources et les instructions de compilation sont disponibles sur le site officiel du projet : http://www.postgresql.org/download/. Ceci étant dit, un serveur FTP anonyme est également mis à la disposition des internautes : ftp://ftp.postgresql.org. Le nom du fichier à télécharger se présente toujours sous la forme « postgresql-X.Y.Z.tar.gz » où X.Y.Z représente la version de PostgreSQL.

Lorsque la future version du logiciel est en phase de démonstration (alpha) ou de test (versions bêta), les sources sont accessibles à ces adresses :

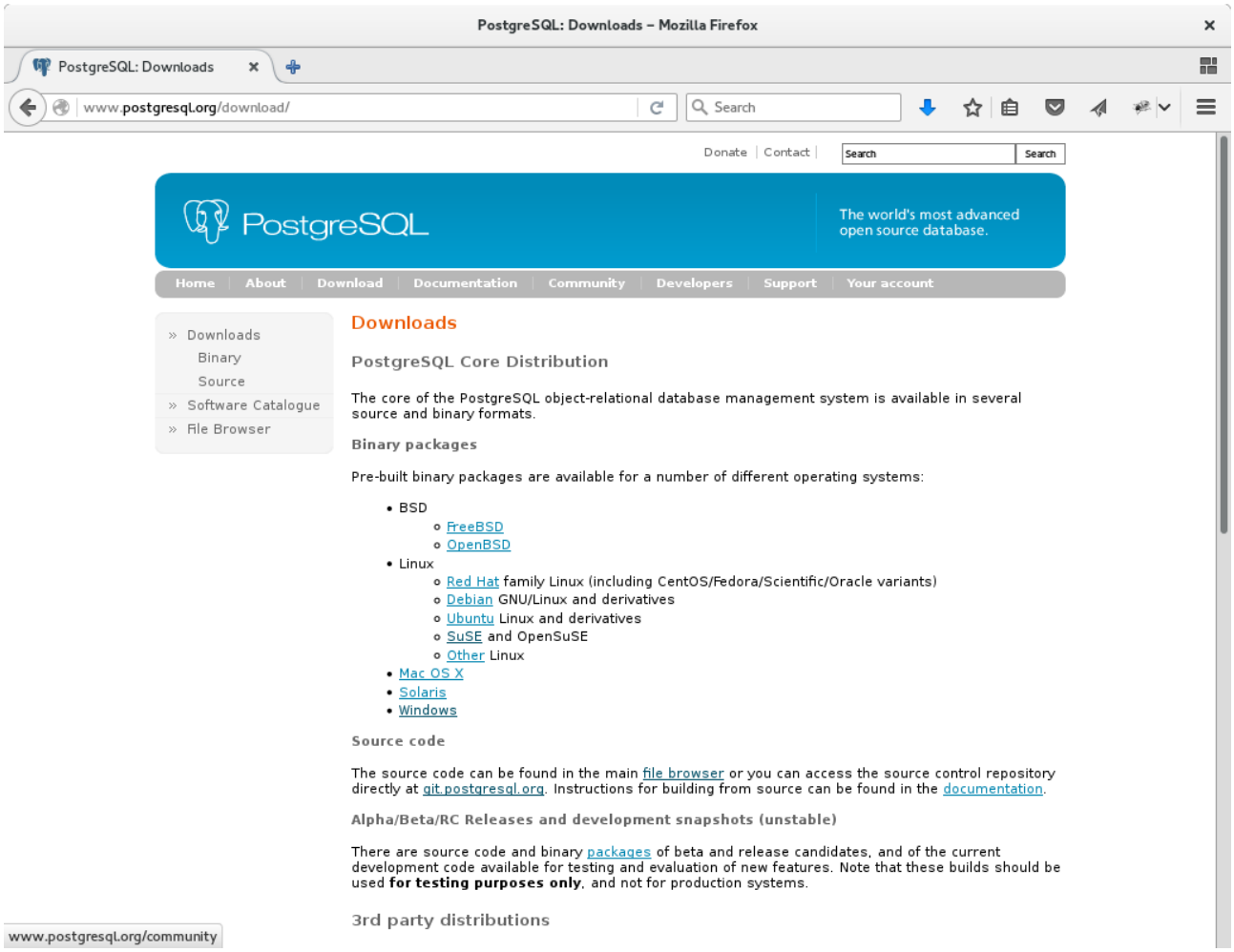
- http://www.postgresql.org/developer/alpha
- http://www.postgresql.org/developer/beta

Voici comment récupérer la dernière version des sources de PostgreSQL :

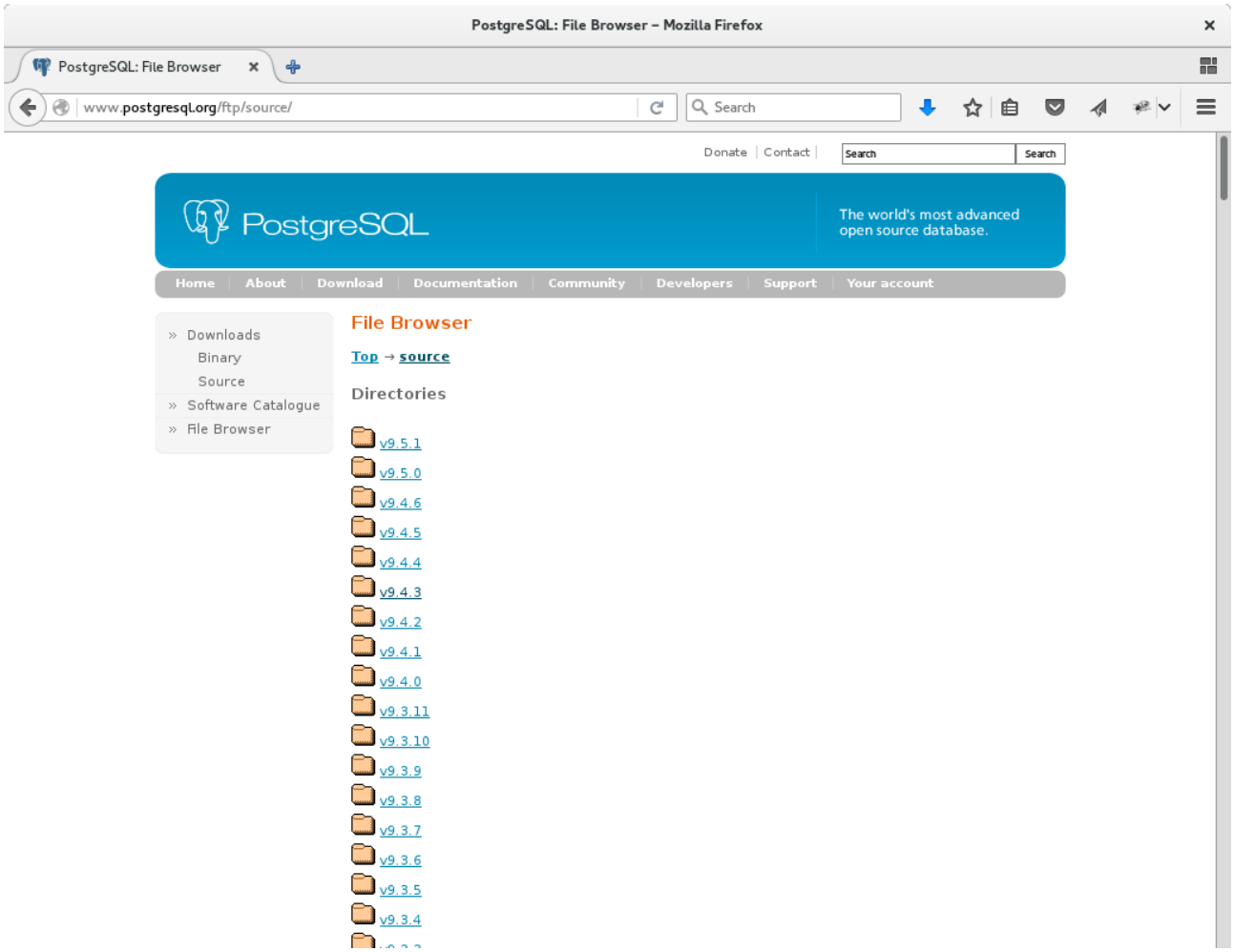
- Aller sur la page d'accueil du projet PostgreSQL



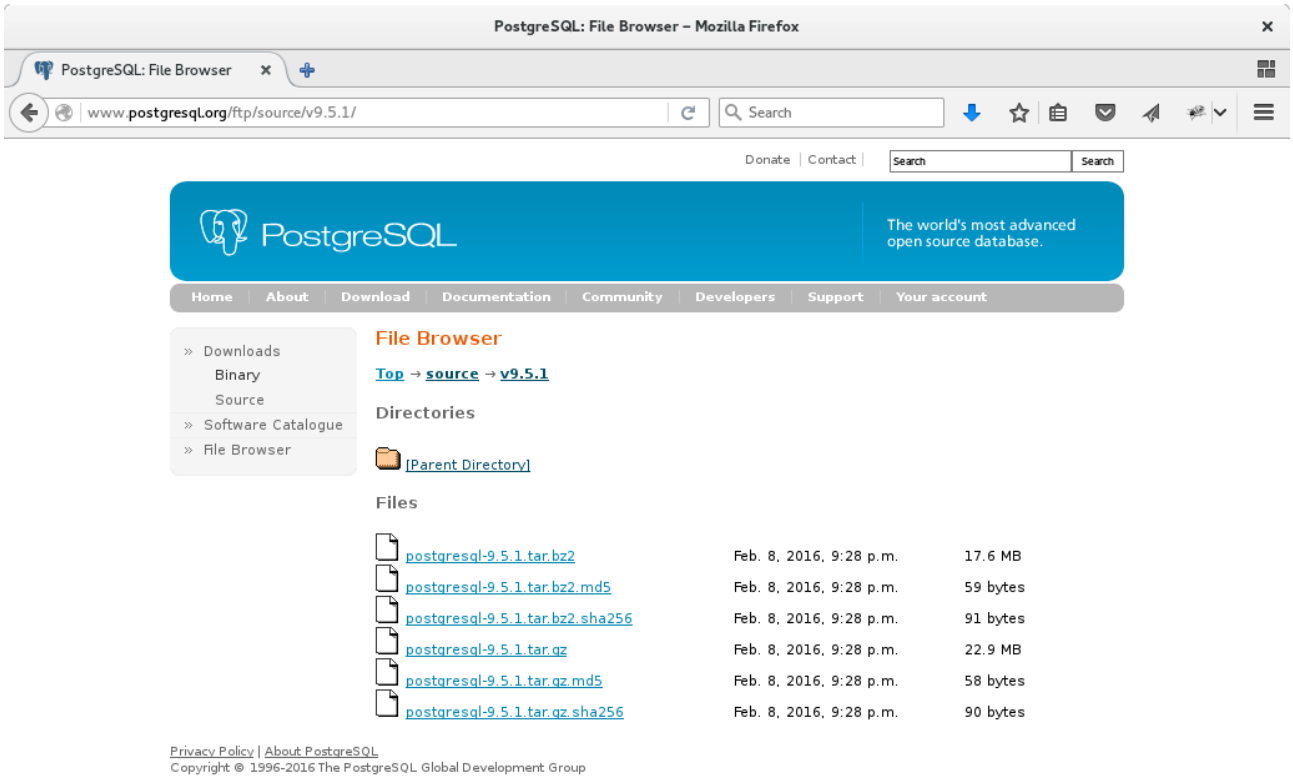

- Cliquer sur le lien « Downloads »



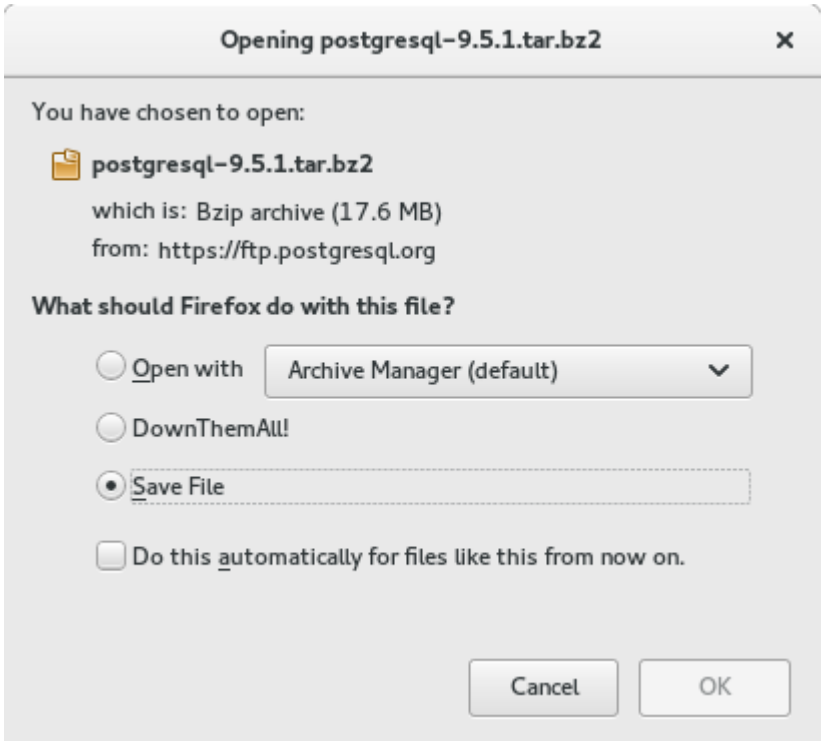

- Cliquer sur le lien « file browser » de la partie « Source code »




- Cliquer sur le lien « v9.5.1 » (la 9.5.1 était la dernière version disponible lors de la mise à jour de cette présentation mais utilisez la version la plus récente possible)




- Cliquer sur le lien « postgresql-9.5.1.tar.bz2 »



## 2.2 Phases de compilation/installation

- Processus standard :

```
$ tar xvfj postgresql-X.Y.Z.tar.bz2
$ cd postgresql-X.Y.Z
$ ./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-X.Y.Z.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-X.Y.Z
$ ./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 :

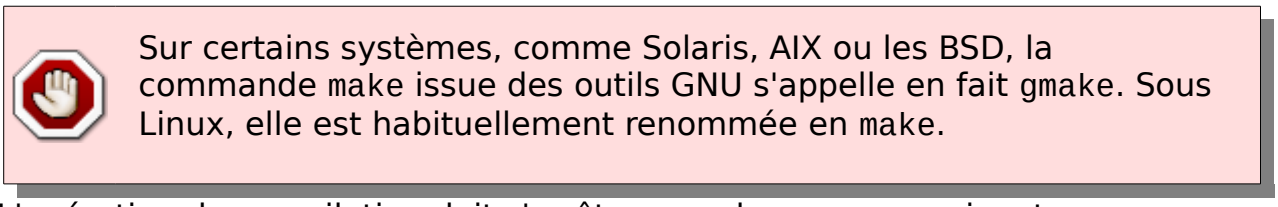
```
configure: creating ./config.status
config.status: creating GNUmakefile
config.status: creating src/Makefile.global
config.status: creating src/include/pg_config.h
config.status: creating src/include/pg_config_ext.h
config.status: creating src/interfaces/ecpg/include/ecpg_config.h
config.status: linking src/backend/port/tas/dummy.s to
src/backend/port/tas.s
config.status: linking src/backend/port/dynloader/linux.c to
src/backend/port/dynloader.c
config.status: linking src/backend/port/sysv_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/backend/port/unix_latch.c to
src/backend/port/pg_latch.c
```

```
config.status: linking src/backend/port/dynloader/linux.h to
src/include/dynloader.h
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
```

On passe alors à 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`.

L'opération de compilation doit s'arrêter avec le message suivant :

```
All of PostgreSQL successfully made. Ready to install.
```

Dans le cas contraire, une erreur s'est produite et il est nécessaire de la corriger avant de continuer.

Si le message de réussite de compilation est affiché, il est possible d'installer le résultat de la compilation :

```
$ make install
```

Là-aussi, un message spécifique doit être affiché pour indiquer le résultat de l'installation :

```
PostgreSQL installation complete.
```

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`.

## 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 a postériori

```
pg_config --configure
```

Le script de configuration de la compilation possède un certain nombre de paramètre 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`: 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 l'interface Perl ainsi que les extensions Perl de 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/8.4/man'
  '--with-docdir=/usr/share/doc/postgresql-doc-8.4'
  '--datadir=/usr/share/postgresql/8.4'
  '--bindir=/usr/lib/postgresql/8.4/bin'
  '--libdir=/usr/lib/postgresql/8.4/lib'
  '--includedir=/usr/include/postgresql/8.4'
  '--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'
```

## 2.4 Tests de 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
  ./pg_regress --inputdir=. --dlpath=. --multibyte=SQL_ASCII  --temp-
install=./tmp_check --top-builddir=../../.. --schedule=./parallel_schedule
  ============== creating temporary installation        ==============
  ============== initializing database system           ==============
  ============== starting postmaster                    ==============
  running on port 57332 with pid 9843
  ============== creating database "regression"         ==============
  CREATE DATABASE
  ALTER DATABASE
  ============== running regression test queries        ==============
  test tablespace                   ... ok
  parallel group (17 tests):  char name text boolean varchar txid int2 int4
oid float4 money float8 uuid int8 enum bit numeric
        boolean                     ... ok
        char                        ... ok
        name                        ... ok
        varchar                     ... ok
        text                        ... ok
  ...
```

Les tests de régression sont une suite de tests qui vérifient que PostgreSQL fonctionne correctement sur la machine cible. Ces tests ne doivent 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.

## 2.5 Création de l'utilisateur

- 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. La version 9.5 rend cette pratique moins intéressante grâce à la possibilité de nommer les instances avec le paramètre `cluster_name`, information affichée au niveau des processus gérant une instance.

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`.

## 2.6 Création du répertoire de données

- Outil initdb
- Création du répertoire

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

- Options d'emplacement
  - `--data` pour les fichiers de données
  - `--xlogdir` pour les journaux de transactions
- Option sécurité
  - `--auth` pour indiquer la méthode d'authentification par défaut
  - `--pwprompt` ou `--pwfile` pour donner un mot de passe à l'utilisateur postgres
  - `--data-checksums` pour ajouter des sommes de contrôle sur les fichier de données

Notez que vous devez exécuter cette commande en étant connecté sous le compte de l'utilisateur PostgreSQL décrit dans la section précédente (généralement l'utilisateur `postgres`).

Si le répertoire n'existe pas, `initdb` tentera de le créer. Toutefois, vous devez veiller à ce qu'il ait les droits pour le faire. S'il existe, il doit être vide.

Voici ce qu'affiche cette commande :

```
$ initdb -D /usr/local/pgsql/data
The files belonging to this database system will be owned by user
"guillaume".
This user must also own the server process.

The database cluster will be initialized with locale "en_US.utf8".
The default database encoding has accordingly been set to "UTF8".
The default text search configuration will be set to "english".

Data page checksums are disabled.

creating directory /home/guillaume/data ... ok
creating subdirectories ... ok
selecting default max_connections ... 100
selecting default shared_buffers ... 128MB
selecting dynamic shared memory implementation ... posix
creating configuration files ... ok
creating template1 database in /home/guillaume/data/base/1 ... ok
initializing pg_authid ... ok
initializing dependencies ... ok
creating system views ... ok
loading system objects' descriptions ... ok
creating collations ... ok
creating conversions ... ok
creating dictionaries ... ok
setting privileges on built-in objects ... ok
creating information schema ... ok
loading PL/pgSQL server-side language ... ok
vacuuming database template1 ... ok
copying template1 to template0 ... ok
copying template1 to postgres ... ok
syncing data to disk ... ok

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:

    /home/guillaume/pg/bin/pg_ctl -D /home/guillaume/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 ;
- création de la base `template1` ;
- ajout des objets système ;
- exécution d'un VACUUM sur cette base ;

- création de la base `template0` à partir de la base `template1` ;
- création de la base `postgres` à partir de la base `template1` ;
- 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`. Il est aussi possible d'activer les sommes de contrôle des fichiers de données avec l'option `--data-checksums`.

## 2.7 Lancement et arrêt

- Script de démarrage

```
# /etc/init.d/postgresql [action]
```

- Outil pg_ctl

```
$ pg_ctl --pgdata /usr/local/pgsql/data -l logfile [action]
```

- `[action]` dépend du besoin
  - start pour démarrer
  - stop pour arrêter
  - reload pour recharger la configuration
  - restart pour redémarrer

La méthode recommandée est d'utiliser le script de démarrage. 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.

L'autre méthode est fonctionnelle mais moins intéressante. Elle est à exécuter avec l'utilisateur qui a été créé précédemment.

Les deux méthodes partagent la majorité des actions présentées ci-dessus. Cependant, `pg_ctl` permet de préciser plus facilement le mode d'arrêt parmi les trois disponibles :

- smart : pour vider le cache de PostgreSQL sur disque, et attendre la fin de l'exécution des clients (pgAdmin, pg_dump, psql, libpq, etc) ;
- fast : 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

(de ce fait, au redémarrage, le serveur devra rejouer ses journaux de transactions).

Par défaut, le mode `fast` est utilisé. Avant la version 9.5, le mode par défaut était `smart`. Il est possible de forcer le mode avec l'option `-m`.

# 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 recommendations officielles de la distribution. Il y a donc des différences, parfois importantes, entre les paquets.

## 3.1 Paquets Debian officiels

- Différents paquets disponibles
  - serveur, client, contrib, docs
  - et les extensions, outils
- `apt-get install postgresql-x.x`
  - 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 par exemple), l'installation de PostgreSQL a été découpée en plusieurs paquets :

- le serveur : postgresql-x.x
- les clients : postgresql-client-x.x
- les modules contrib : postgresql-contrib-x.x

- la documentation : postgresql-doc-x.x

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-x.x
- PL/perl dans postgresql-plperl-x.x
- PL/tcl dans postgresql-pltcl-x.x
- 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-x.x.

Le `x.x` correspond à la version majeure souhaitée (9.5 par exemple). 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/lib/postgresql/x.x/bin`, les fichiers de configuration sont dans `/etc/postgresql/x.x/instance`, les traces dans `/var/log/postgresql/postgresql-x.x.log`, les données dans `/var/lib/postgresql/x.x/instance`. `instance` correspond au nom de l'instance (`main` par défaut). Ceci est fait pour respecter le plus possible la norme FHS (`Filesystem Hierarchy Standard`, http://www.pathname.com/fhs/).

Les wrappers sont des scripts écrit par les mainteneurs de paquets Debian pour faciliter la création, la suppression et la gestion de différentes instances sur le même serveur.

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), création du répertoire des données, lancement du serveur. 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.

## 3.2 Paquets Debian communautaires

- La communauté met des paquets Debian à disposition :
  - http://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/ jessie-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 direct 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 prévenu trois/quatre jours avant la sortie d'une version de son imminence 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`. On ajoutera la déclaration suivante dans les fichiers de configuration du gestionnaire `apt`, par exemple pour une distribution Debian 8 (nom de code Jessie) :

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

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

```
lsb_release -c
```

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-get update
```

## 3.3 Paquets RedHat 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 RedHat et les versions dérivées (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

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.

Le xx correspond à la version majeure souhaitée (95 pour une version 9.5 par exemple). 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-x.x/bin`, les traces dans

`/var/lib/pgsql/x.x/data/pg_log` (utilisation du `logger process` de PostgreSQL), les données dans `/var/lib/pgsql/x.x/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`), RedHat 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-x.x/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 on fournit l'action `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-x.x /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/x.x/serv2
PORT=5433
```

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

## 3.4 Paquets RedHat communautaires

- La communauté met des paquets RedHat à disposition :
  - http://yum.postgresql.org
- Synchrone avec le projet PostgreSQL
- Ajout du dépôt grâce aux paquets RPM disponibles

La différence entre les paquets officiels et ceux de la communauté tient principalement en une disponibilité immédiate des mises à jour.

L'installation de la configuration du dépôt est très simple. Il suffit de télécharger le paquet RPM de définition du dépôt, puis de l'installer avec la commande `rpm`.

# 4 Installation sous Windows

- Disponible pour les différentes versions NT de Windows (2003 server, 2008 server, 2012 server, etc).
- Installeur graphique proposé par EnterpriseDB
- Ou archive des binaires

Le portage de PostgreSQL sous Windows date de la version 8.0. Lors du développement de cette version, le travail qu'il a nécessité fût suffisamment important pour justifier à lui seul le passage de la branche 7 à la branche 8 du projet. La version Windows a été considérée comme une version beta pendant les versions 8.0 et 8.1. La version 8.2 est donc la première version stable de PostgreSQL sous Windows. Seules les versions NT sont supportés car seules ces versions disposent du système de fichiers NTFS. Ce système de fichiers est obligatoire car, contrairement à la VFAT, il gère les liens symboliques (appelés jonctions sous Windows). Une version 64 bits n'est disponible que depuis la version 9.0.

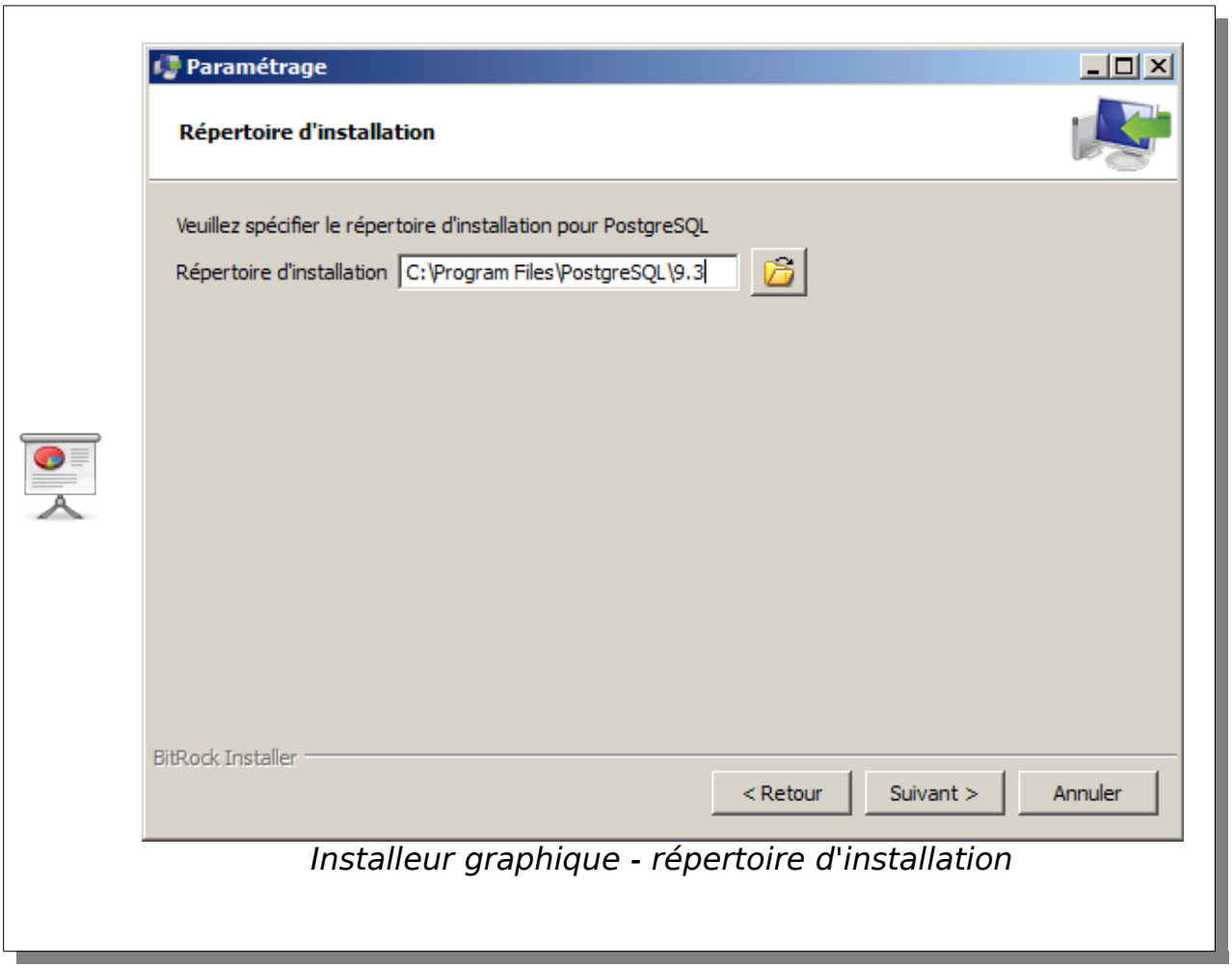
## 4.1 Installeur graphique - bienvenue


*Installeur graphique - bienvenue*

C'est l'écran d'accueil de l'installeur, il suffit de cliquer sur le bouton Suivant.

L'indication « Packaged by EnterpriseDB » signifie que l'installeur a été développé et proposé au téléchargement par la société EnterpriseDB. En effet, maintenir un installeur Windows est un travail conséquent et personne dans la communauté ne souhaite actuellement faire ce travail. Il n'empêche que cet installeur est disponible gratuitement et propose la version communautaire de PostgreSQL.

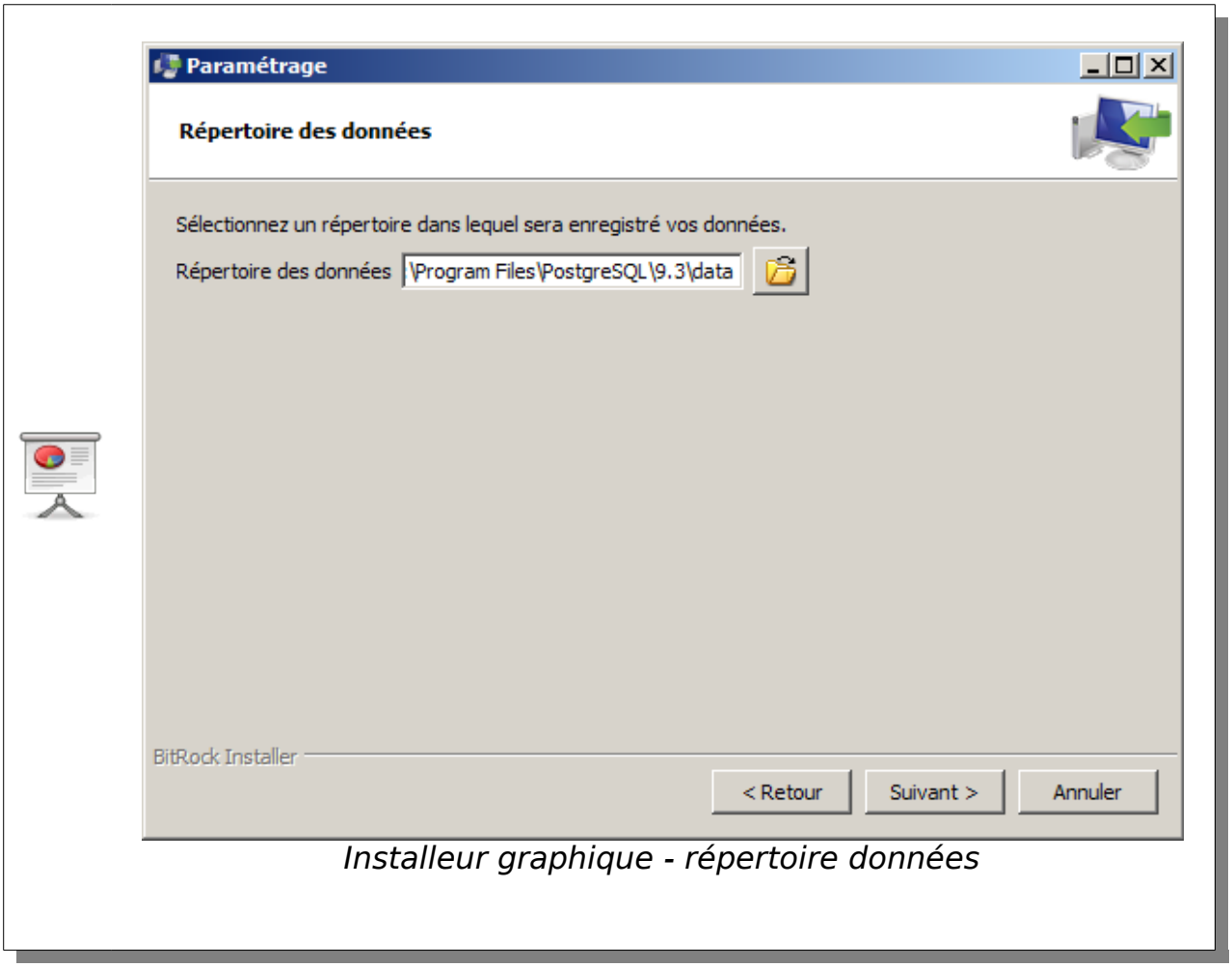
## 4.2 Installeur graphique - répertoire d'installation




*Installeur graphique - répertoire d'installation*

Cet écran sert à saisir le répertoire d'installation. Ce dernier a une valeur par défaut généralement convenable car il n'existe pas vraiment de raison d'installer les binaires PostgreSQL en dehors du répertoire Programmes.

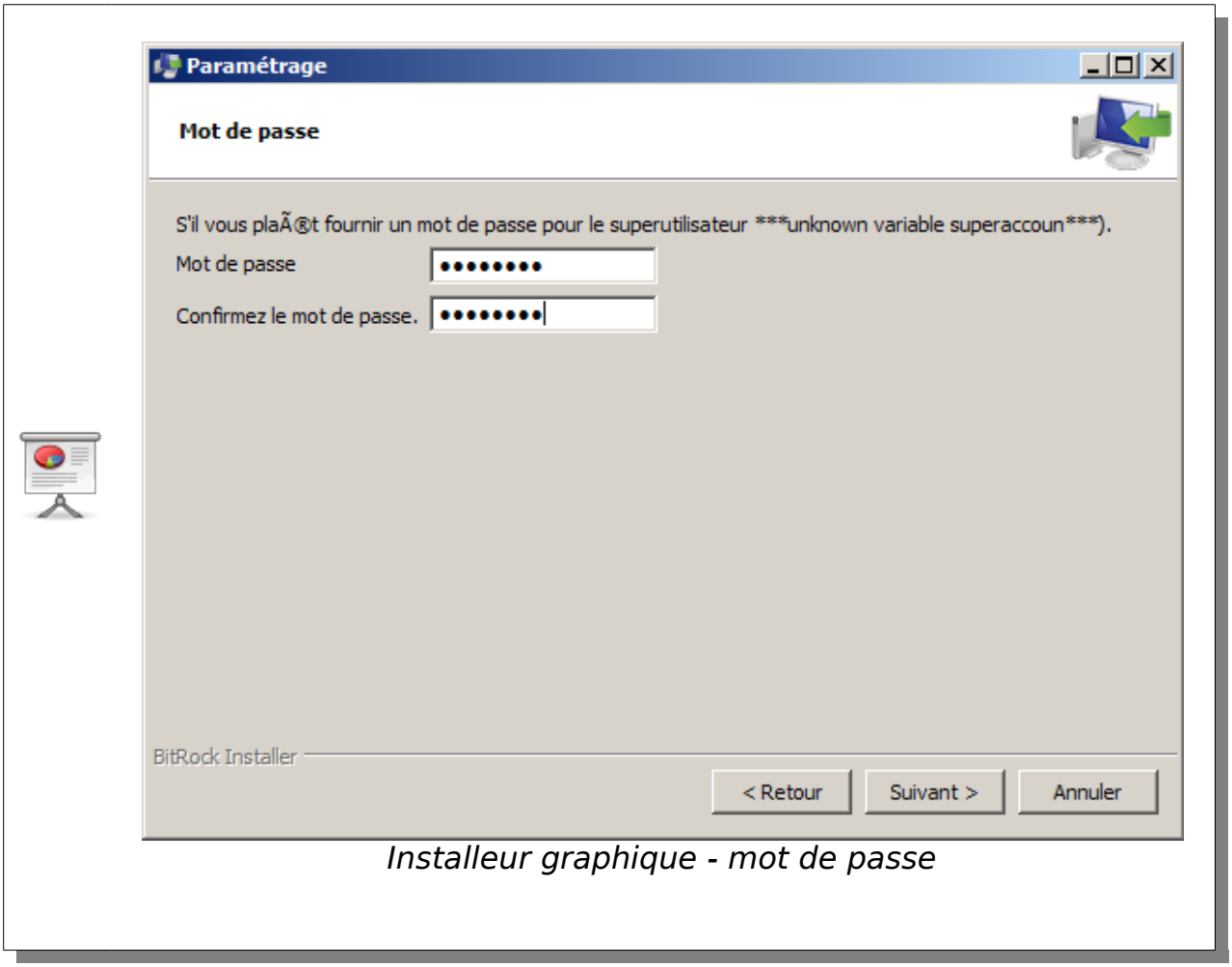
## 4.3 Installeur graphique - répertoire données




*Installeur graphique - répertoire données*

Cet écran sert à saisir le répertoire des données de l'instance PostgreSQL. La valeur par défaut de ce dernier ne convient pas forcément à toutes les installations. Il est par exemple souvent modifié pour que le répertoire des données se trouve sur un autre disque que le disque système.

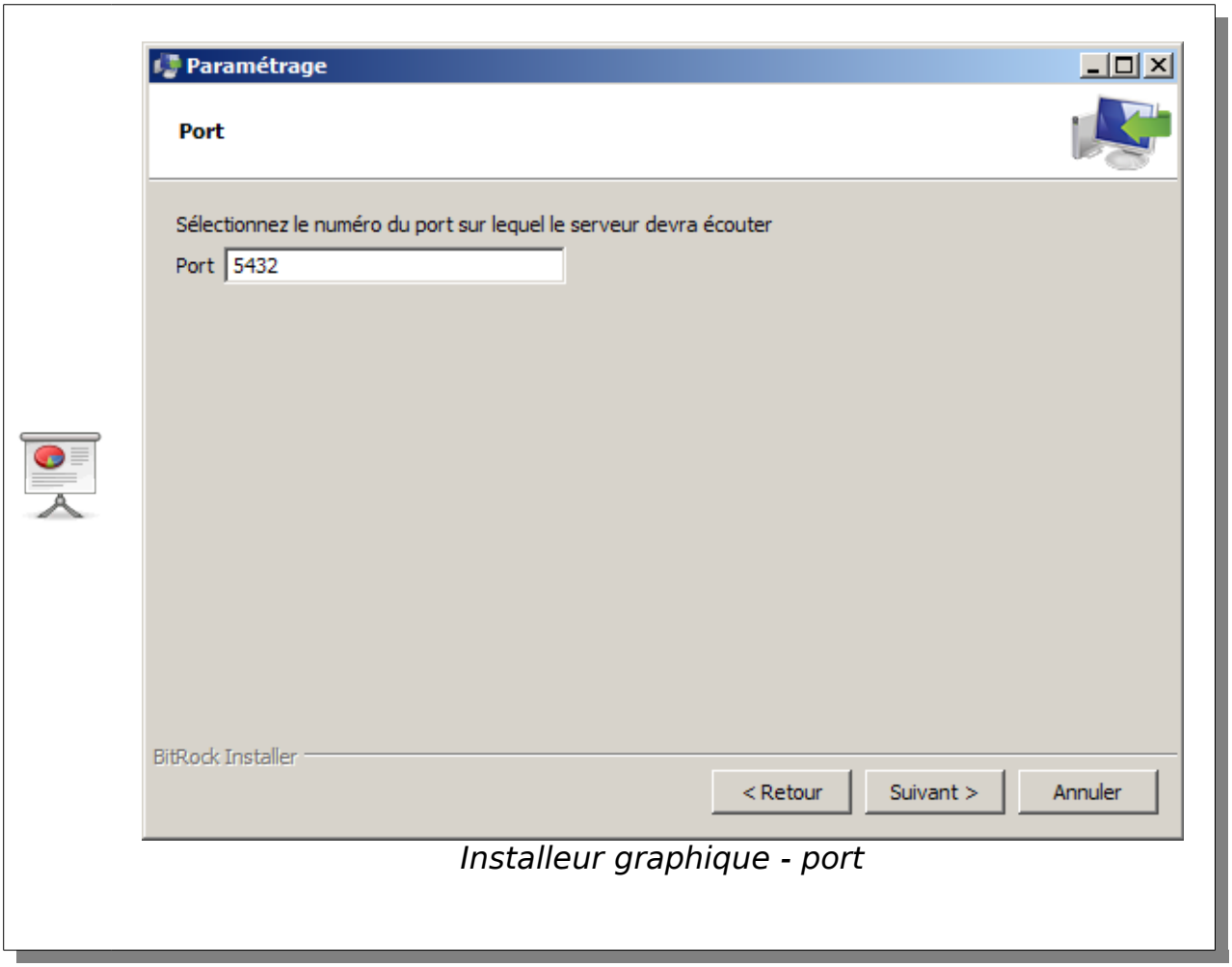
## 4.4 Installeur graphique - mot de passe




*Installeur graphique - mot de passe*

L'installeur graphique crée un compte utilisateur système. Le mot de passe saisi ici est le mot de passe de ce compte système. Il sera aussi utilisé par le compte de l'utilisateur `postgres` sous PostgreSQL. En cas de mise à jour, il faut saisir l'ancien mot de passe.

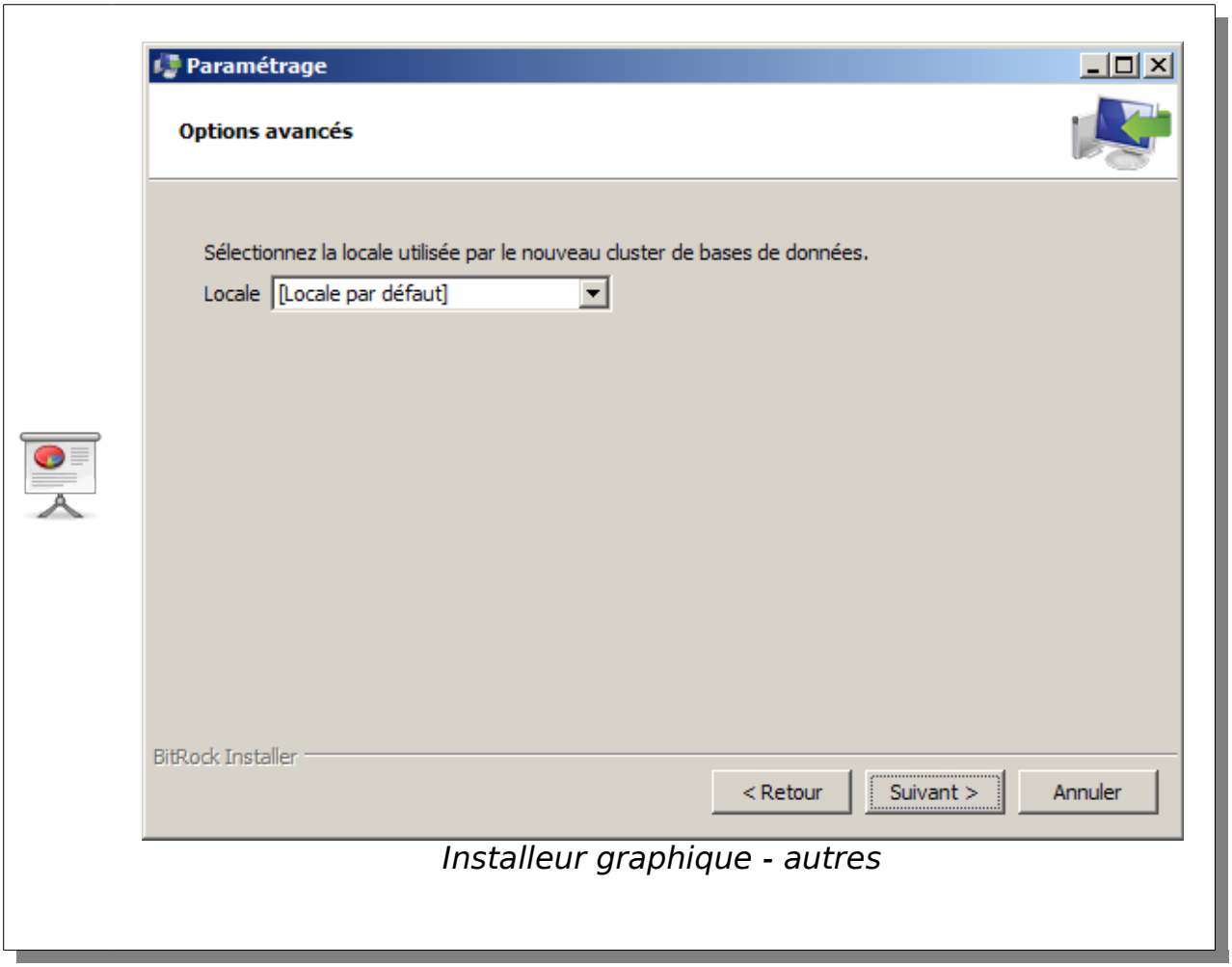
## 4.5 Installeur graphique - port




*Installeur graphique - port*

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

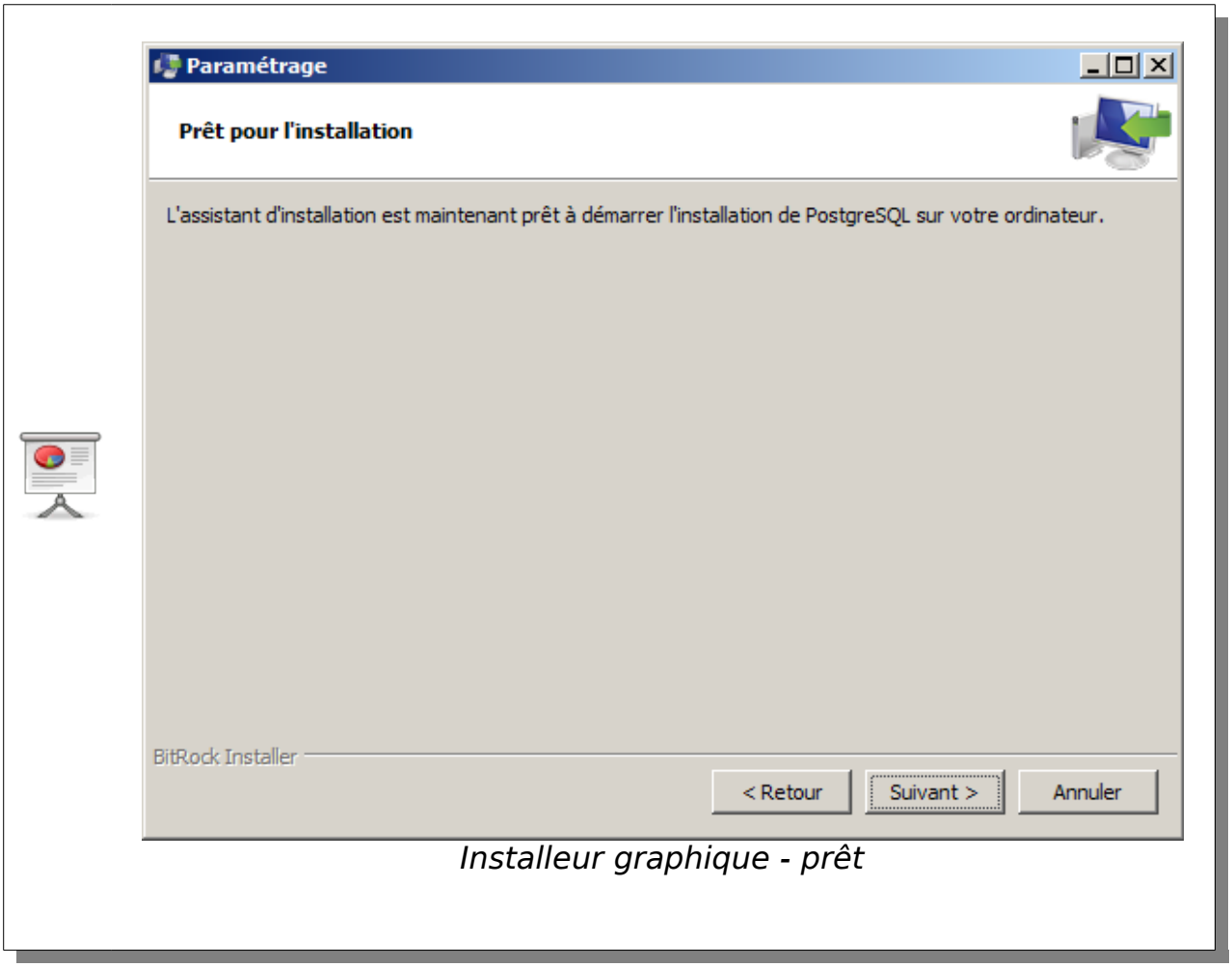
## 4.6 Installeur graphique - autres




*Installeur graphique - autres*

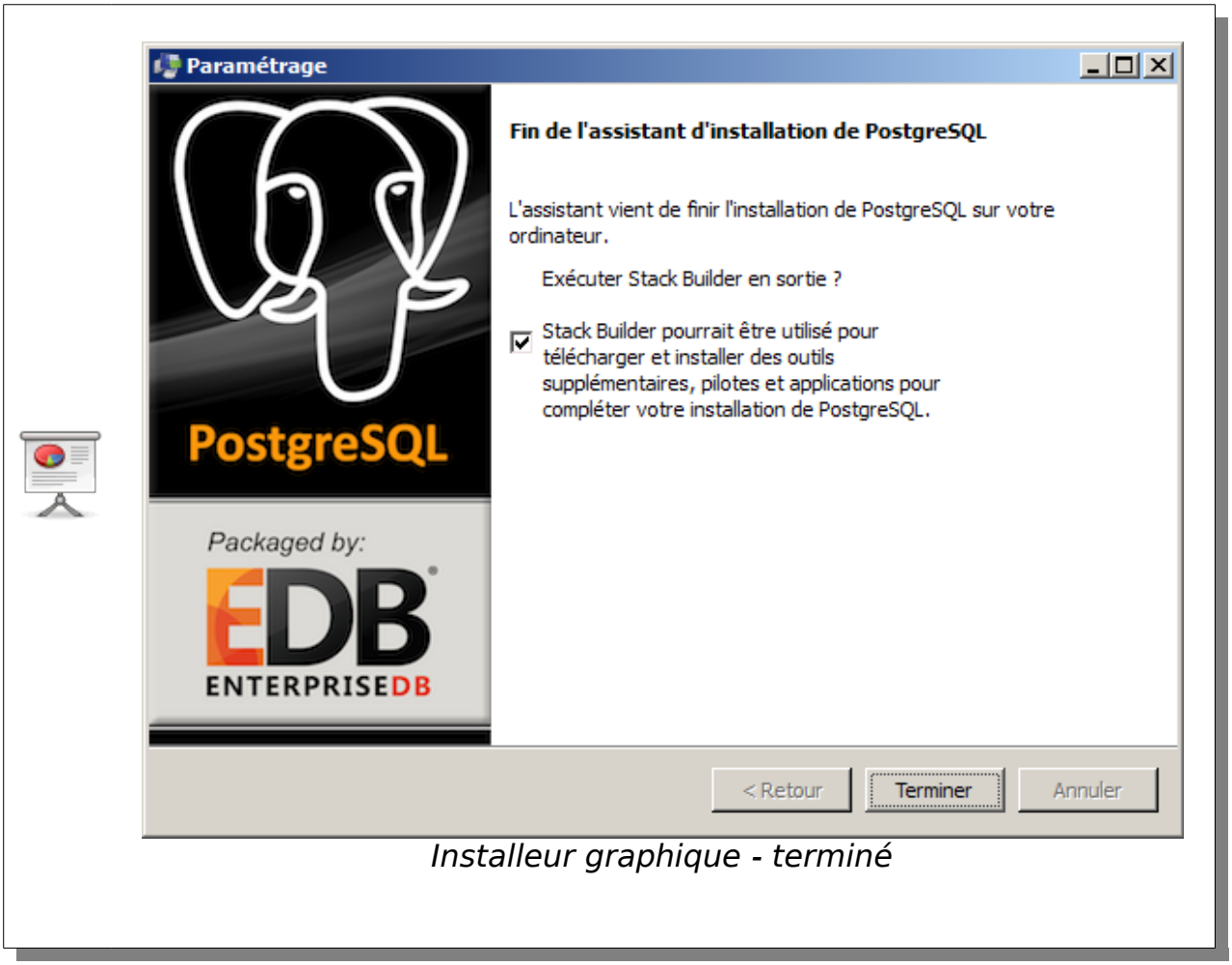
L'installeur propose d'utiliser la locale par défaut mais il est possible de la modifier.

## 4.7 Installeur graphique - prêt




*Installeur graphique - prêt*

## 4.8 Installeur graphique - terminé




*Installeur graphique - terminé*

À ce moment-là, un nouvel utilisateur a été créé avec le nom `postgres`. La commande `initdb` a été exécutée pour créer le répertoire des données. Un service a été 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. Il est à noter que cet installeur a aussi installé pgAdmin, dans une version compatible avec la version du serveur PostgreSQL.

Cet installeur existe aussi sous Mac OS X et sous Linux. Autant il est préférable de passer par les paquets de sa distribution Linux, autant il est recommandé d'utiliser cet installeur Mac OS X.

# 5 Premiers réglages

- Sécurité
- Configuration minimale
- Démarrage
- Test de connexion

## 5.1 Sécurité

- Politique de mot de passe
  - 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.

Autant il est possible de ne pas fournir de mot de passe pour l'utilisateur système, autant il faut absolument en donner un pour le rôle `postgres`. Cela se fait avec cette requête :

```
ALTER ROLE postgres ENCRYPTED PASSWORD 'mot de passe';
```

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 si applicable, en utilisant quel rôle, pour quelle base de données et avec quel méthode d'authentification.

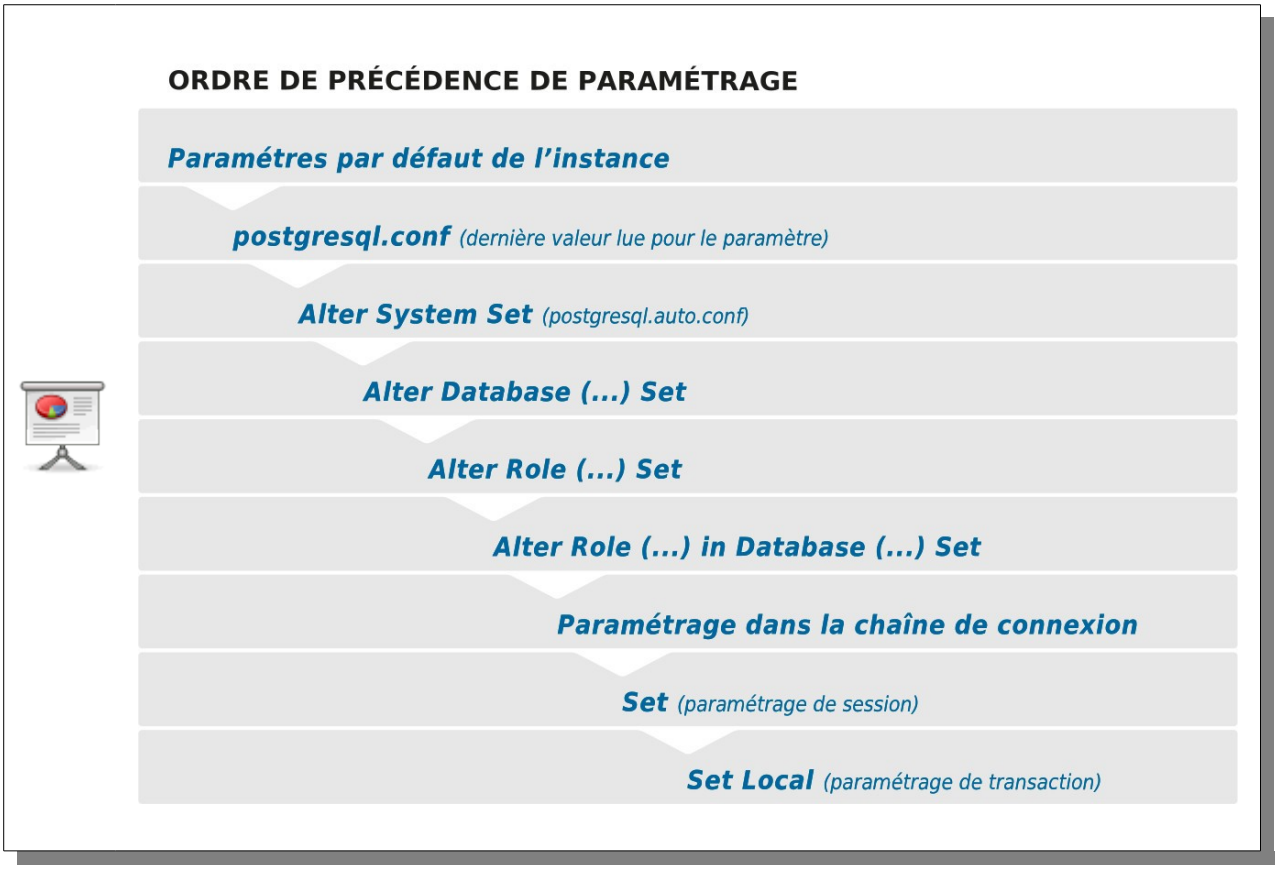
## 5.2 Configuration minimale

- Fichier `postgresql.conf`
- Configuration du moteur
- Plus de 200 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 un sous-répertoire du répertoire `/etc`.

Ce fichier contient beaucoup de paramètres, plus de 200, mais seuls quelques-uns sont essentiels à connaître pour avoir une instance fiable et performante.

## 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`.

## 5.4 Configuration des connexions


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.

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.

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 que chaque processus ait accès à plus de mémoire) ou de l'augmenter (pour qu'un plus grand nombre d'utilisateurs puisse se connecter en même temps). Attention toutefois à ne pas dépasser 1000 car cela serait source de contre-performance.

## 5.5 Configuration des ressources mémoire

- shared_buffers
- work_mem
- maintenance_work_mem

Chaque fois que PostgreSQL a besoin de lire ou 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`. Sur un serveur dédié à PostgreSQL, une valeur d'un quart de la mémoire vive totale est généralement suffisante. Suivant les cas, une valeur inférieure ou supérieure sera encore meilleure pour les performances. À vous de tester ce qui vous convient le mieux. Attention, il faut noter que l'augmentation de cette valeur peut nécessiter la modification du paramètre système SHMMAX pour que PostgreSQL ait le droit de réclamer autant de mémoire au démarrage. Cette restriction disparaît avec PostgreSQL 9.3.

`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 et hachage, alors que `maintenance_work_mem` est la taille de la mémoire pour les `VACUUM`, `CREATE INDEX` et ajout de clé étrangère. Cette mémoire est distincte de celle allouée par le paramètre `shared_buffers`, et elle sera allouée pour chaque nouvelle connexion PostgreSQL. Donc, si votre `max_connexions` 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.

## 5.6 Configuration des journaux de transactions 1/2

- Paramètre : fsync

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. Le paramètre `fsync` permet de désactiver l'envoi de l'ordre de synchronisation au système d'exploitation. Ce paramètre ne doit jamais être désactivé en production.

## 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 écrit dans le cache “système de fichier” du système d'exploitation (OS) situé en RAM (mémoire volatile). Celui-ci confirme au processus que l'écriture a bien été réalisée. Le bloc sera écrit sur disque “plus tard” afin de grouper les demandes d'écritures des autres processus et réduire les déplacements des têtes de lecture/écriture des disques qui sont des opérations coûteuses en temps. Entre la confirmation et l'écriture réelle sur les disques il peut se passer un certain délai, si une panne arrivait durant celui-ci les données n'étant pas encore présentes physiquement sur les disques elles seraient donc perdues.
- Dans le cas d'une écriture synchrone : Un processus écrit dans le cache du système d'exploitation et demande à l'OS d'effectuer une synchronisation sur disque. Le bloc est donc écrit physiquement sur les disques et le processus a la confirmation de l'écriture à la fin de l'opération. Il a donc la garantie que la donnée est bien présente physiquement sur les disques. Cette opération est très coûteuse et lente (déplacement des têtes des disques). Pour gagner en performance, les constructeurs ont donc mis en place un système de cache dans les cartes RAID, le processus a la confirmation de l'écriture une fois la donnée présente dans ce cache même si 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.

## 5.8 Configuration des traces

- log_destination
- logging_collector
- log_line_prefix
- lc_messages

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 « log collector » qui va enregistrer les messages dans des fichiers. Ce démon s'active en configurant le paramètre `logging_collector` à `on`. Si vous faites cela, n'oubliez pas de changer le préfixe des messages pour ajouter au moins l'horodatage (`log_line_prefix = '%t'` ). Des traces sans date et heure ne servent à rien.

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 les outils pgFouine et pgBadger, pourtant écrit par des Français, ne savent pas interpréter des traces en français. De plus, la moindre recherche sur Internet ramènera plus de liens avec des traces en anglais.

## 5.9 Configuration des démons

- 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.

# 6 Mise à jour

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

## 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 on trouve des conseils essentiels.

Les « Releases Notes » sont présentes dans l'annexe E de la documentation officielle (http://docs.postgresql.fr/current/release.html).

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 donnée, sauf cas exceptionnel qui serait alors précisé dans les notes de version (par exemple, de la 8.1.14 à la 8.1.15, il est nécessaire de reconstruire les index de type GiST). Pensez éventuellement à faire une sauvegarde préalable par sécurité.

A contrario, si la mise à jour consiste en un changement de version majeure (par exemple, de la 9.2 à la 9.4), 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.

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

## 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-get` sur Debian et affiliés, `yum` sur RedHat 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.

## 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
- Pour mettre à jour
  - mise à jour des binaires
  - et mise à jour/traitement des fichiers de données
- 3 méthodes
  - dump/restore
  - Slony
  - 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.

Une fois les tests 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.

## 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 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 est conséquent (plus d'une centaine de Go) ou si les autres méthodes ne sont pas possibles.

## 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 qu'il est acté que le nouveau serveur donne pleine satisfaction, il suffit de désinstaller Slony des deux côtés.

## 6.6 Mise à jour majeure par pg_upgrade

- Outil 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.

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.

# 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 !

## 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 :
http://docs.postgresql.fr/current/installation.html

ainsi que la partie spécifique à l'installation sur :
http://docs.postgresql.fr/current/INSTALL.html

Les documentations de Dalibo pour l'installation de PostgreSQL sur Windows sont disponibles sur :

- http://www.dalibo.org/installation_de_postgresql_8.3_sous_windows
- http://www.dalibo.org/installation_de_postgresql_8.4_sous_windows
- http://www.dalibo.org/installation_de_postgresql_9.0_sous_windows

## 7.2 Questions

N'hésitez pas, c'est le moment !