Module M1

Architecture de PostgreSQL

25.03

Table des matières

Sur ce document . . . 1
Chers lectrices & lecteurs, . . . 1
À propos de DALIBO . . . 1
Remerciements . . . 2
Forme de ce manuel . . . 2
Licence Creative Commons CC-BY-NC-SA . . . 2
Marques déposées . . . 3
Versions de PostgreSQL couvertes . . . 3

1/ Architecture & fichiers de PostgreSQL 5
1.1 Au menu . . . 6
1.2 Rappels sur l'installation . . . 7
1.2.1 Paquets précompilés . . . 7
1.2.2 Installons PostgreSQL . . . 8
1.3 Processus de PostgreSQL . . . 9
1.3.1 Introduction . . . 9
1.3.2 Processus d'arrière-plan . . . 10
1.3.3 Processus d'arrière-plan (suite) . . . 11
1.4 Processus par client (client backend) . . . 13
1.5 Gestion de la mémoire . . . 15
1.6 Fichiers . . . 16
1.6.1 Répertoire de données . . . 17
1.6.2 Fichiers de configuration . . . 18
1.6.3 Autres fichiers dans PGDATA . . . 18
1.6.4 Fichiers de données . . . 20
1.6.5 Fichiers liés aux transactions . . . 22
1.6.6 Fichiers liés à la réplication . . . 24
1.6.7 Répertoire des tablespaces . . . 25
1.6.8 Fichiers des statistiques d'activité . . . 26
1.6.9 Autres répertoires . . . 27
1.6.10 Les fichiers de traces (journaux) . . . 27
1.7 Résumé . . . 28
1.8 Conclusion . . . 29
1.8.1 Questions . . . 29
1.9 Quiz . . . 30
1.10 Installation de PostgreSQL depuis les paquets communautaires . . . 31
1.10.1 Sur Rocky Linux 8 ou 9 . . . 31
1.10.2 Sur Debian / Ubuntu . . . 34
1.10.3 Accès à l'instance depuis le serveur même (toutes distributions) . . . 36
1.11 Travaux pratiques . . . 38
1.11.1 Processus . . . 38
1.11.2 Fichiers . . . 39

1.12 Travaux pratiques (solutions) 41
1.12.1 Processus 41
1.12.2 Fichiers 44

Les formations Dalibo **51**
Cursus des formations 51
Les livres blancs 52
Téléchargement gratuit 52

Sur ce document

Formation	Module M1
Titre	Architecture de PostgreSQL
Révision	25.03
PDF	https://dali.bo/m1_pdf
EPUB	https://dali.bo/m1_epub
HTML	https://dali.bo/m1_html
Slides	https://dali.bo/m1_slides
TP	https://dali.bo/m1_tp
TP (solutions)	https://dali.bo/m1_solutions

Vous trouverez en ligne les différentes versions complètes de ce document.

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[1] !

À propos de DALIBO

DALIBO est le spécialiste français de PostgreSQL. Nous proposons du support, de la formation et du conseil depuis 2005.

Retrouvez toutes nos formations sur https://dalibo.com/formations

[1] mailto:formation@dalibo.com

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 :

Alexandre Anriot, Jean-Paul Argudo, Carole Arnaud, Alexandre Baron, David Bidoc, Sharon Bonan, Franck Boudehen, Arnaud Bruniquel, Pierrick Chovelon, Damien Clochard, Christophe Courtois, Marc Cousin, Gilles Darold, Ronan Dunklau, Vik Fearing, Stefan Fercot, Dimitri Fontaine, Pierre Giraud, Nicolas Gollet, Florent Jardin, Virginie Jourdan, Luc Lamarle, Denis Laxalde, Guillaume Lelarge, Alain Lesage, Benoit Lobréau, Jean-Louis Louër, Thibaut Madelaine, Adrien Nayrat, Alexandre Pereira, Flavie Perette, Robin Portigliatti, Thomas Reiss, Maël Rimbault, Jehan-Guillaume de Rorthais, Julien Rouhaud, Stéphane Schildknecht, Julien Tachoires, Nicolas Thauvin, Be Hai Tran, Christophe Truffier, Arnaud de Vathaire, Cédric Villemain, Thibaud Walkowiak, Frédéric Yhuel.

Forme de ce manuel

Les versions PDF, EPUB ou HTML de ce document sont structurées autour des slides de nos formations. Le texte suivant chaque slide contient le cours et de nombreux détails qui ne peuvent être données à l'oral.

Licence Creative Commons CC-BY-NC-SA

Cette formation est sous licence **CC-BY-NC-SA**[2]. Vous êtes libre de la redistribuer et/ou modifier aux conditions suivantes :

- Paternité
- Pas d'utilisation commerciale
- Partage des conditions initiales à l'identique

Vous n'avez pas le droit d'utiliser cette création à des fins commerciales.

Si vous modifiez, transformez ou adaptez cette création, vous n'avez le droit de distribuer la création qui en résulte que sous un contrat identique à celui-ci.

Vous devez citer le nom de l'auteur original de la manière indiquée par l'auteur de l'œuvre ou le titulaire des droits qui vous confère cette autorisation (mais pas d'une manière qui suggérerait qu'ils vous soutiennent ou approuvent votre utilisation de l'œuvre). À chaque réutilisation ou distribution de cette création, vous devez faire apparaître clairement au public les conditions contractuelles de sa mise à disposition. La meilleure manière de les indiquer est un lien vers cette page web. Chacune de ces conditions peut être levée si vous obtenez l'autorisation du titulaire des droits sur cette œuvre. Rien dans ce contrat ne diminue ou ne restreint le droit moral de l'auteur ou des auteurs.

Le texte complet de la licence est disponible sur http://creativecommons.org/licenses/by-nc-sa/2.0/fr/legalcode

[2] http://creativecommons.org/licenses/by-nc-sa/2.0/fr/legalcode

Cela inclut les diapositives, les manuels eux-mêmes et les travaux pratiques. Cette formation peut également contenir quelques images et schémas dont la redistribution est soumise à des licences différentes qui sont alors précisées.

Marques déposées

PostgreSQL® Postgres® et le logo Slonik sont des marques déposées[3] par PostgreSQL Community Association of Canada.

Versions de PostgreSQL couvertes

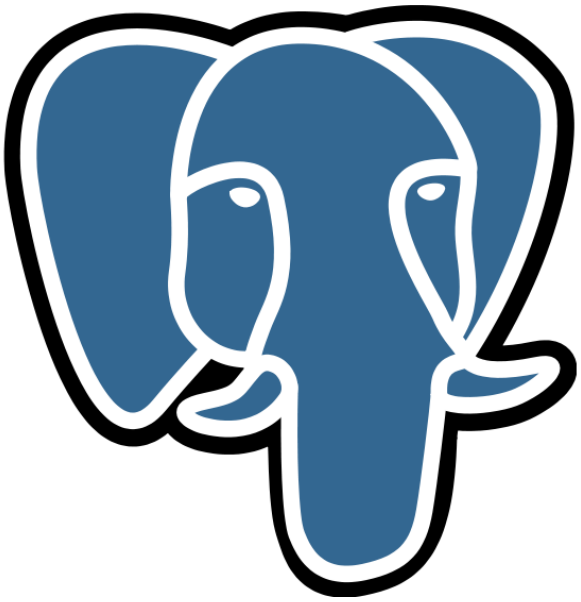
Ce document ne couvre que les versions supportées de PostgreSQL au moment de sa rédaction, soit les versions 13 à 17.

Sur les versions précédentes susceptibles d'être encore rencontrées en production, seuls quelques points très importants sont évoqués, en plus éventuellement de quelques éléments historiques.

Sauf précision contraire, le système d'exploitation utilisé est Linux.

[3] https://www.postgresql.org/about/policies/trademarks/

1/ Architecture & fichiers de PostgreSQL



1.1 AU MENU

- Rappels sur l'installation
- Les processus
- Les fichiers

Le présent module vise à donner un premier aperçu global du fonctionnement interne de PostgreSQL.

Après quelques rappels sur l'installation, nous verrons essentiellement les processus et les fichiers utilisés.

1.2 RAPPELS SUR L'INSTALLATION

- Plusieurs possibilités
  - paquets Linux précompilés
  - outils externes d'installation
  - code source
- Chacun ses avantages et inconvénients
  - Dalibo recommande fortement les paquets précompilés

Nous recommandons très fortement l'utilisation des paquets Linux précompilés. Dans certains cas, il ne sera pas possible de faire autrement que de passer par des outils externes, comme l'installeur d'EntrepriseDB sous Windows.

1.2.1 Paquets précompilés

- Paquets Debian ou Red Hat suivant la distribution utilisée
- Préférence forte pour ceux de la communauté
- Installation du paquet
  - installation des binaires
  - création de l'utilisateur postgres
  - initialisation d'une instance (Debian seulement)
  - lancement du serveur (Debian seulement)
- (Red Hat) Script de création de l'instance

Debian et Red Hat fournissent des paquets précompilés adaptés à leur distribution. Dalibo recommande d'installer les paquets de la communauté, ces derniers étant bien plus à jour que ceux des distributions.

L'installation d'un paquet provoque la création d'un utilisateur système nommé postgres et l'installation des binaires. Suivant les distributions, l'emplacement des binaires change. Habituellement, tout est placé dans `/usr/pgsql-<version majeure>` pour les distributions Red Hat et dans `/usr/lib/postgresql/<version majeure>` pour les distributions Debian.

Dans le cas d'une distribution Debian, une instance est immédiatement créée dans `/var/lib/postgresql/<version`
Elle est ensuite démarrée.

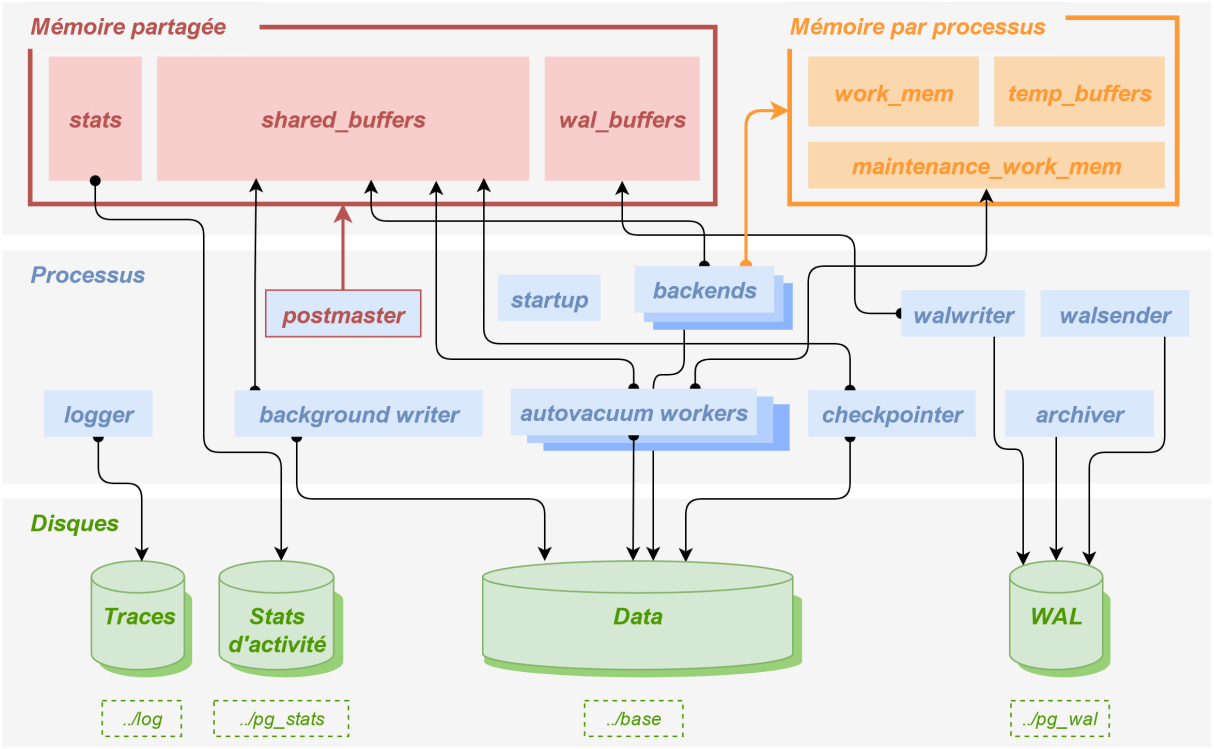
Dans le cas d'une distribution Red Hat, aucune instance n'est créée automatiquement. Il faudra utiliser un script (dont le nom dépend de la version de la distribution) pour créer l'instance, puis nous pourrons utiliser le script de démarrage pour lancer le serveur.

1.2.2 Installons PostgreSQL

- Prenons un moment pour
  - installer PostgreSQL
  - créer une instance
  - démarrer l'instance
- Pas de configuration spécifique pour l'instant

L'annexe ci-dessous décrit l'installation de PostgreSQL sans configuration particulière pour suivre le reste de la formation.

1.3 PROCESSUS DE POSTGRESQL



1.3.1 Introduction

- PostgreSQL est :
  - multiprocessus (et non multithread)
  - à mémoire partagée
  - client-serveur

L'architecture PostgreSQL est une architecture multiprocessus et non multithread.

Cela signifie que chaque processus de PostgreSQL s'exécute dans un contexte mémoire isolé, et que la communication entre ces processus repose sur des mécanismes systèmes inter-processus : sémaphores, zones de mémoire partagée, sockets. Ceci s'oppose à l'architecture multithread, où l'ensemble du moteur s'exécute dans un seul processus, avec plusieurs threads (contextes) d'exécution, où tout est partagé par défaut.

Le principal avantage de cette architecture multiprocessus est la stabilité : un processus, en cas de problème, ne corrompt que sa mémoire (ou la mémoire partagée), le plantage d'un processus n'affecte

pas directement les autres. Son principal défaut est une allocation statique des ressources de mémoire partagée : elles ne sont pas redimensionnables à chaud.

Tous les processus de PostgreSQL accèdent à une zone de « mémoire partagée ». Cette zone contient les informations devant être partagées entre les clients, comme un cache de données, ou des informations sur l'état de chaque session par exemple.

PostgreSQL utilise une architecture client-serveur. Nous ne nous connectons à PostgreSQL qu'à travers d'un protocole bien défini, nous n'accédons jamais aux fichiers de données.

1.3.2 Processus d'arrière-plan

```
# ps f -e --format=pid,command | grep -E "postgres|postmaster"
  96122 /usr/pgsql-17/bin/postmaster -D /var/lib/pgsql/15/data/
  96123  \_ postgres: logger
  96125  \_ postgres: checkpointer
  96126  \_ postgres: background writer
  96127  \_ postgres: walwriter
  96128  \_ postgres: autovacuum launcher
  96131  \_ postgres: logical replication launcher
```

(sous Rocky Linux 8)

Nous constatons que plusieurs processus sont présents dès le démarrage de PostgreSQL. Nous allons les détailler.

NB : sur Debian, le postmaster est nommé *postgres* comme ses processus fils ; sous Windows les noms des processus sont par défaut moins verbeux.

1.3.3 Processus d'arrière-plan (suite)

- Les processus présents au démarrage :
  - Un processus père : `postmaster`
  - `background writer`
  - `checkpointer`
  - `walwriter`
  - `autovacuum launcher`
  - `stats collector` (avant v15)
  - `logical replication launcher`
- et d'autres selon la configuration et le moment :
  - dont les *background workers* : parallélisation, extensions…

Le `postmaster` est responsable de la supervision des autres processus, ainsi que de la prise en compte des connexions entrantes.

Le `background writer` et le `checkpointer` s'occupent d'effectuer les écritures en arrière plan, évitant ainsi aux sessions des utilisateurs de le faire.

Le `walwriter` écrit le journal de transactions de façon anticipée, afin de limiter le travail de l'opération `COMMIT`.

L'`autovacuum launcher` pilote les opérations d'« autovacuum ».

Avant la version 15, le `stats collector` collecte les statistiques d'activité du serveur. À partir de la version 15, ces informations sont conservées en mémoire jusqu'à l'arrêt du serveur où elles sont stockées sur disque jusqu'au prochain démarrage.

Le `logical replication launcher` est un processus dédié à la réplication logique.

Des processus supplémentaires peuvent apparaître, comme un `walsender` dans le cas où la base est le serveur primaire du cluster de réplication, un `logger` si PostgreSQL doit gérer lui-même les fichiers de traces (par défaut sous Red Hat, mais pas sous Debian), ou un `archiver` si l'instance est paramétrée pour générer des archives de ses journaux de transactions.

Ces différents processus seront étudiées en détail dans d'autres modules de formation.

Aucun de ces processus ne traite de requête pour le compte des utilisateurs. Ce sont des processus d'arrière-plan effectuant des tâches de maintenance.

Il existe aussi les *background workers* (processus d'arrière-plan), lancés par PostgreSQL, mais aussi par des extensions tierces. Par exemple, la parallélisation des requêtes se base sur la création tem-

poraire de *background workers* épaulant le processus principal de la requête. La réplication logique utilise des *background workers* à plus longue durée de vie. De nombreuses extensions en utilisent pour des raisons très diverses. Le paramètre `max_worker_processes` régule le nombre de ces *workers*. Ne descendez pas en-dessous du défaut (8). Il faudra même parfois monter plus haut.

1.4 PROCESSUS PAR CLIENT (CLIENT BACKEND)

- Pour chaque client, nous avons un processus :
  - créé à la connexion
  - dédié au client…
  - …et qui dialogue avec lui
  - détruit à la déconnexion
- Un processus gère une requête
  - peut être aidé par d'autres processus
- Le nombre de processus est régi par les paramètres :
  - `max_connections` (défaut : 100) - connexions réservées
  - compromis nombre requêtes actives/nombre cœurs/complexité/mémoire

Pour chaque nouvelle session à l'instance, le processus `postmaster` crée un processus fils qui s'occupe de gérer cette session. Il y a donc un processus dédié à chaque connexion cliente, et ce processus est détruit à fin de cette connexion.

Ce processus dit *backend* reçoit les ordres SQL, les interprète, exécute les requêtes, trie les données, et enfin retourne les résultats. Dans certains cas, il peut demander le lancement d'autres processus pour l'aider dans l'exécution d'une requête en lecture seule (parallélisme).

Le dialogue entre le client et ce processus respecte un protocole réseau bien défini. Le client n'a jamais accès aux données par un autre moyen que par ce protocole.

Le nombre maximum de connexions à l'instance simultanées, actives ou non, est limité par le paramètre `max_connections`. Le défaut est 100.

Certaines connexions sont réservées. Les administrateurs doivent pouvoir se connecter à l'instance même si la limite est atteinte. Quelques connexions sont donc réservées aux superutilisateurs (paramètre `superuser_reserved_connections`, à 3 par défaut). On peut aussi octroyer le rôle `pg_use_reserved_connections` à certains utilisateurs pour leur garantir l'accès à un nombre de connexions réservées, à définir avec le paramètre `reserved_connections` (vide par défaut), cela à partir de PostreSQL 16.

Attention : modifier un de ces paramètres impose un redémarrage complet de l'instance, puisqu'ils ont un impact sur la taille de la mémoire partagée entre les processus PostgreSQL. Il faut donc réfléchir au bon dimensionnement avant la mise en production.

La valeur 100 pour `max_connections` est généralement suffisante. Il peut être intéressant de la diminuer pour se permettre de monter `work_mem` et autoriser plus de mémoire de tri. Il est possible de

monter `max_connections` pour qu'un plus grand nombre de clients puisse se connecter en même temps.

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. Ce dimensionnement est encore compliqué par le parallélisme et la limitation de la bande passante des disques.

Intercaler un « pooler » 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.

Le plus réputé est PgBouncer, mais il est aussi souvent inclus dans des serveurs d'application (par exemple Tomcat).

1.5 GESTION DE LA MÉMOIRE

Structure de la mémoire sous PostgreSQL

- Zone de mémoire partagée :
  - *shared buffers* surtout
  - …
- Zone de chaque processus
  - tris en mémoire (`work_mem`)
  - …

La gestion de la mémoire dans PostgreSQL mérite un module de formation à lui tout seul.

Pour le moment, bornons-nous à la séparer en deux parties : la mémoire partagée et celle attribuée à chacun des nombreux processus.

La mémoire partagée stocke principalement le cache des données de PostgreSQL (*shared buffers*, paramètre `shared_buffers`), et d'autres zones plus petites : cache des journaux de transactions, données de sessions, les verrous, etc.

La mémoire propre à chaque processus sert notamment aux tris en mémoire (définie en premier lieu par le paramètre `work_mem`), au cache de tables temporaires, etc.

1.6 FICHIERS

- Une instance est composée de fichiers :
  - Répertoire de données
  - Fichiers de configuration
  - Fichier PID
  - Tablespaces
  - Statistiques
  - Fichiers de trace

Une instance est composée des éléments suivants :

Le répertoire de données :

Il contient les fichiers obligatoires au bon fonctionnement de l'instance : fichiers de données, journaux de transaction….

Les fichiers de configuration :

Selon la distribution, ils sont stockés dans le répertoire de données (Red Hat et dérivés comme CentOS ou Rocky Linux), ou dans `/etc/postgresql` (Debian et dérivés).

Un fichier PID :

Il permet de savoir si une instance est démarrée ou non, et donc à empêcher un second jeu de processus d'y accéder.

Le paramètre `external_pid_file` permet d'indiquer un emplacement où PostgreSQL créera un second fichier de PID, généralement à l'extérieur de son répertoire de données.

Des tablespaces :

Ils sont totalement optionnels. Ce sont des espaces de stockage supplémentaires, stockés habituellement dans d'autres systèmes de fichiers.

Le fichier de statistiques d'exécution :

Généralement dans `pg_stat_tmp/`.

Les fichiers de trace :

Typiquement, des fichiers avec une variante du nom `postgresql.log`, souvent datés. Ils sont par défaut dans le répertoire de l'instance, sous `log/`. Sur Debian, ils sont redirigés vers la sortie d'erreur du système. Ils peuvent être redirigés vers un autre mécanisme du système d'exploitation (syslog sous Unix, journal des événements sous Windows),

1.6.1 Répertoire de données

```
postgres$ ls $PGDATA
base                pg_ident.conf  pg_stat      pg_xact
current_logfiles    pg_logical     pg_stat_tmp  postgresql.auto.conf
global              pg_multixact   pg_subtrans  postgresql.conf
log                 pg_notify      pg_tblspc    postmaster.opts
pg_commit_ts        pg_replslot    pg_twophase  postmaster.pid
pg_dynshmem         pg_serial      PG_VERSION
pg_hba.conf         pg_snapshots   pg_wal
```

- Une seule instance PostgreSQL doit y accéder !

Le répertoire de données est souvent appelé PGDATA, du nom de la variable d'environnement que l'on peut faire pointer vers lui pour simplifier l'utilisation de nombreux utilitaires PostgreSQL. Il est possible aussi de le connaître, une fois connecté à une base de l'instance, en interrogeant le paramètre `data_directory`.

```
SHOW data_directory;

      data_directory
---------------------------
 /var/lib/pgsql/15/data
```

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 (ou depuis un conteneur comme docker).
Faites donc bien attention de ne lancer PostgreSQL qu'une seule fois sur un répertoire de données.

Il est recommandé de ne jamais créer ce répertoire PGDATA à la racine d'un point de montage, quel que soit le système d'exploitation et le système de fichiers utilisé. Si un point de montage est dédié à l'utilisation de PostgreSQL, positionnez-le toujours dans un sous-répertoire, voire deux niveaux en dessous du point de montage. (par exemple `<point de montage>/<version majeure>/<nom instance>`).

Voir à ce propos le chapitre *Use of Secondary File Systems* dans la documentation officielle : https://www.postgresql.org/docs/current/creating-cluster.html.

Vous pouvez trouver une description de tous les fichiers et répertoires dans la documentation officielle[1].

[1] https://www.postgresql.org/docs/current/static/storage-file-layout.html

1.6.2 Fichiers de configuration

- `postgresql.conf` (+ fichiers inclus)
- `postgresql.auto.conf`
- `pg_hba.conf` (+ fichiers inclus (v16))
- `pg_ident.conf` (idem)

Les fichiers de configuration sont de simples fichiers textes. Habituellement, ce sont les suivants.

`postgresql.conf` contient une liste de paramètres, sous la forme `paramètre=valeur`. Tous les paramètres sont modifiables (et présents) dans ce fichier. Selon la configuration, il peut inclure d'autres fichiers, mais ce n'est pas le cas par défaut. Sous Debian, il est sous `/etc`.

`postgresql.auto.conf` stocke les paramètres de configuration fixés en utilisant la commande `ALTER SYSTEM`. Il surcharge donc `postgresql.conf`. Comme pour `postgresql.conf`, sa modification impose de recharger la configuration ou redémarrer l'instance. Il est techniquement possible, mais déconseillé, de le modifier à la main. `postgresql.auto.conf` est toujours dans le répertoire des données, même si `postgresql.conf` est ailleurs.

`pg_hba.conf` contient les règles d'authentification à la base selon leur identité, la base, la provenance, etc.

`pg_ident.conf` est plus rarement utilisé. Il complète `pg_hba.conf`, par exemple pour rapprocher des utilisateurs système ou propres à PostgreSQL.

Ces deux derniers fichiers peuvent eux-mêmes inclure d'autres fichiers (depuis PostgreSQL 16). Leur utilisation est décrite dans notre première formation[2].

1.6.3 Autres fichiers dans PGDATA

- `PG_VERSION` : fichier contenant la version majeure de l'instance
- `postmaster.pid`
  - nombreuses informations sur le processus père
  - fichier externe possible, paramètre `external_pid_file`
- `postmaster.opts`

[2] https://dali.bo/f_html

`PG_VERSION` est un fichier. Il contient en texte lisible la version majeure devant être utilisée pour accéder au répertoire (par exemple `15`). On trouve ces fichiers `PG_VERSION` à de nombreux endroits de l'arborescence de PostgreSQL, par exemple dans chaque répertoire de base du répertoire `PGDATA/base/` ou à la racine de chaque tablespace.

Le fichier `postmaster.pid` est créé au démarrage de PostgreSQL. PostgreSQL y indique le PID du processus père sur la première ligne, l'emplacement du répertoire des données sur la deuxième ligne et la date et l'heure du lancement de postmaster sur la troisième ligne ainsi que beaucoup d'autres informations. Par exemple :

```
~$ cat /var/lib/postgresql/15/data/postmaster.pid
7771
/var/lib/postgresql/15/data
1503584802
5432
/tmp
localhost
  5432001  54919263
ready

$ ps -HFC postgres
UID  PID    SZ     RSS PSR STIME TIME   CMD
pos 7771 0 42486 16536   3 16:26 00:00  /usr/local/pgsql/bin/postgres
                                          -D /var/lib/postgresql/15/data
pos 7773 0 42486  4656   0 16:26 00:00  postgres: checkpointer
pos 7774 0 42486  5044   1 16:26 00:00  postgres: background writer
pos 7775 0 42486  8224   1 16:26 00:00  postgres: walwriter
pos 7776 0 42850  5640   1 16:26 00:00  postgres: autovacuum launcher
pos 7777 0 42559  3684   0 16:26 00:00  postgres: logical replication launcher

$ ipcs -p |grep  7771
54919263   postgres   7771       10640

$ ipcs  | grep 54919263
0x0052e2c1 54919263   postgres   600        56         6
```

Le processus père de cette instance PostgreSQL a comme PID le 7771. Ce processus a bien réclamé une sémaphore d'identifiant 54919263. Cette sémaphore correspond à des segments de mémoire partagée pour un total de 56 octets. Le répertoire de données se trouve bien dans `/var/lib/postgresql/15/data`.

Le fichier `postmaster.pid` est supprimé lors de l'arrêt de PostgreSQL. Cependant, ce n'est pas le cas après un arrêt brutal. Dans ce genre de cas, PostgreSQL détecte le fichier et indique qu'il va malgré tout essayer de se lancer s'il ne trouve pas de processus en cours d'exécution avec ce PID. Un fichier supplémentaire peut être créé ailleurs grâce au paramètre `external_pid_file`, c'est notamment le défaut sous Debian :

```
external_pid_file = '/var/run/postgresql/15-main.pid'
```

Par contre, ce fichier ne contient que le PID du processus père.

Quant au fichier `postmaster.opts`, il contient les arguments en ligne de commande correspondant au dernier lancement de PostgreSQL. Il n'est jamais supprimé. Par exemple :

```
$ cat $PGDATA/postmaster.opts
/usr/local/pgsql/bin/postgres "-D" "/var/lib/postgresql/15/data"
```

1.6.4 Fichiers de données

- `base/` : contient les fichiers de données
  - un sous-répertoire par base de données
  - `pgsql_tmp` : fichiers temporaires
- `global/` : contient les objets globaux à toute l'instance

`base/` contient les fichiers de données (tables, index, vues matérialisées, séquences). Il contient un sous-répertoire par base, le nom du répertoire étant l'OID de la base dans `pg_database`. Dans ces répertoires, nous trouvons un ou plusieurs fichiers par objet à stocker. Ils sont nommés ainsi :

- Le nom de base du fichier correspond à l'attribut `relfilenode` de l'objet stocké, dans la table `pg_class` (une table, un index…). Il peut changer dans la vie de l'objet (par exemple lors d'un `VACUUM FULL`, un `TRUNCATE`…)
- Si le nom est suffixé par un « . » suivi d'un chiffre, il s'agit d'un fichier d'extension de l'objet : un objet est découpé en fichiers de 1 Go maximum.
- Si le nom est suffixé par `_fsm`, il s'agit du fichier stockant la *Free Space Map* (liste des blocs réutilisables).
- Si le nom est suffixé par `_vm`, il s'agit du fichier stockant la *Visibility Map* (liste des blocs intégralement visibles, et donc ne nécessitant pas de traitement par `VACUUM`).

Un fichier `base/1247/14356.1` est donc le second segment de l'objet ayant comme `relfilenode` 14356 dans le catalogue `pg_class`, dans la base d'OID 1247 dans la table `pg_database`.

Savoir identifier cette correspondance ne sert que dans des cas de récupération de base très endommagée. Vous n'aurez jamais, durant une exploitation normale, besoin d'obtenir cette correspondance. Si, par exemple, vous avez besoin de connaître la taille de la table `test` dans une base, il vous suffit d'exécuter la fonction `pg_table_size()`. En voici un exemple complet :

```
CREATE TABLE test (id integer);
INSERT INTO test SELECT generate_series(1, 5000000);
SELECT pg_table_size('test');

 pg_table_size
---------------
     181305344
```

Depuis la ligne de commande, il existe un utilitaire nommé `oid2name`, dont le but est de faire la liaison entre le nom de fichier et le nom de l'objet PostgreSQL. Il a besoin de se connecter à la base :

```
$ pwd
/var/lib/pgsql/15/data/base/16388

$ /usr/pgsql-17/bin/oid2name -f 16477 -d employes
From database "employes":
  Filenode         Table Name
----------------------------
     16477  employes_big_pkey
```

Le répertoire `base` peut aussi contenir un répertoire `pgsql_tmp`. Ce répertoire contient des fichiers temporaires utilisés pour stocker les résultats d'un tri ou d'un hachage. À partir de la version 12, il est possible de connaître facilement le contenu de ce répertoire en utilisant la fonction `pg_ls_tmpdir()`, ce qui peut permettre de suivre leur consommation.

Si nous demandons au sein d'une première session :

```
SELECT * FROM generate_series(1,1e9) ORDER BY random() LIMIT 1 ;
```

alors nous pourrons suivre les fichiers temporaires depuis une autre session :

```
SELECT * FROM pg_ls_tmpdir() ;
```

```
       name         |    size    |      modification
--------------------+------------+------------------------
 pgsql_tmp12851.16 | 1073741824 | 2020-09-02 15:43:27+02
 pgsql_tmp12851.11 | 1073741824 | 2020-09-02 15:42:32+02
 pgsql_tmp12851.7  | 1073741824 | 2020-09-02 15:41:49+02
 pgsql_tmp12851.5  | 1073741824 | 2020-09-02 15:41:29+02
 pgsql_tmp12851.9  | 1073741824 | 2020-09-02 15:42:11+02
 pgsql_tmp12851.0  | 1073741824 | 2020-09-02 15:40:36+02
 pgsql_tmp12851.14 | 1073741824 | 2020-09-02 15:43:06+02
 pgsql_tmp12851.4  | 1073741824 | 2020-09-02 15:41:19+02
 pgsql_tmp12851.13 | 1073741824 | 2020-09-02 15:42:54+02
 pgsql_tmp12851.3  | 1073741824 | 2020-09-02 15:41:09+02
 pgsql_tmp12851.1  | 1073741824 | 2020-09-02 15:40:47+02
 pgsql_tmp12851.15 | 1073741824 | 2020-09-02 15:43:17+02
 pgsql_tmp12851.2  | 1073741824 | 2020-09-02 15:40:58+02
 pgsql_tmp12851.8  | 1073741824 | 2020-09-02 15:42:00+02
 pgsql_tmp12851.12 | 1073741824 | 2020-09-02 15:42:43+02
 pgsql_tmp12851.10 | 1073741824 | 2020-09-02 15:42:21+02
 pgsql_tmp12851.6  | 1073741824 | 2020-09-02 15:41:39+02
 pgsql_tmp12851.17 |  546168976 | 2020-09-02 15:43:32+02
```

Le répertoire `global/` contient notamment les objets globaux à toute une instance, comme la table des bases de données, celle des rôles ou celle des tablespaces ainsi que leurs index.

1.6.5 Fichiers liés aux transactions

- `pg_wal/` : journaux de transactions
  - sous-répertoire `archive_status`
  - nom : *timeline*, journal, segment
  - ex : `00000002 00000142 000000FF`
  - créés, initialisés à zéro, recyclés
- `pg_xact/` : état des transactions
- mais aussi : `pg_commit_ts/`, `pg_multixact/`, `pg_serial/` `pg_snapshots/`, `pg_subtrans/`, `pg_twophase/`
- **Ces fichiers sont vitaux !**

Le répertoire `pg_wal` contient les journaux de transactions. Ces journaux garantissent la durabilité des données dans la base, en traçant toute modification devant être effectuée **AVANT** de l'effectuer réellement en base.

Les fichiers contenus dans `pg_wal` ne doivent **jamais** être effacés manuellement. Ces fichiers sont cruciaux au bon fonctionnement de la base. PostgreSQL gère leur création et suppression. S'ils sont toujours présents, c'est que PostgreSQL en a besoin.

Par défaut, les fichiers des journaux font tous 16 Mo. Ils ont des noms sur 24 caractères, comme par exemple :

```
$ ls -l
total 2359320
...
-rw------- 1 postgres postgres 33554432 Mar 26 16:28 00000002000001420000007C
-rw------- 1 postgres postgres 33554432 Mar 26 16:28 00000002000001420000007D
-rw------- 1 postgres postgres 33554432 Mar 26 16:28 00000002000001420000007E
-rw------- 1 postgres postgres 33554432 Mar 26 16:28 00000002000001420000007F
-rw------- 1 postgres postgres 33554432 Mar 26 16:28 00000002000001430000000
-rw------- 1 postgres postgres 33554432 Mar 26 16:28 00000002000001430000001
-rw------- 1 postgres postgres 33554432 Mar 26 16:28 00000002000001430000002
-rw------- 1 postgres postgres 33554432 Mar 26 16:28 00000002000001430000003
...
-rw------- 1 postgres postgres 33554432 Mar 26 16:28 00000002000001430000001D
-rw------- 1 postgres postgres 33554432 Mar 26 16:28 00000002000001430000001E
-rw------- 1 postgres postgres 33554432 Mar 26 16:28 00000002000001430000001F
-rw------- 1 postgres postgres 33554432 Mar 26 16:28 00000002000001430000020
-rw------- 1 postgres postgres 33554432 Mar 26 16:25 00000002000001430000021
-rw------- 1 postgres postgres 33554432 Mar 26 16:25 00000002000001430000022
```

```
-rw------- 1 postgres postgres 33554432 Mar 26 16:25 000000020000014300000023
-rw------- 1 postgres postgres 33554432 Mar 26 16:25 000000020000014300000024
drwx------ 2 postgres postgres    16384 Mar 26 16:28 archive_status
```

La première partie d'un nom de fichier (ici `00000002`) correspond à la *timeline* (« ligne de temps »), qui ne s'incrémente que lors d'une restauration de sauvegarde ou une bascule entre serveurs primaire et secondaire. La deuxième partie (ici `00000142` puis `00000143`) correspond au numéro de journal à proprement parler, soit un ensemble de fichiers représentant 4 Go. La dernière partie correspond au numéro du segment au sein de ce journal. Selon la taille du segment fixée à l'initialisation, il peut aller de `00000000` à `000000FF` (256 segments de 16 Mo, configuration par défaut, soit 4 Go), à `00000FFF` (4096 segments de 1 Mo), ou à `0000007F` (128 segments de 32 Mo, exemple ci-dessus), etc. Une fois ce maximum atteint, le numéro de journal au centre est incrémenté et les numéros de segments reprennent à zéro.

L'ordre d'écriture des journaux est numérique (en hexadécimal), et leur archivage doit suivre cet ordre. Il ne faut pas se fier à la date des fichiers pour le tri : pour des raisons de performances, PostgreSQL recycle généralement les fichiers en les renommant. Dans l'exemple ci-dessus, le dernier journal écrit est `000000020000014300000020` et non `000000020000014300000024`.

Toujours pour des raisons de performance liées aux métadata (voir cette explication d'Andres Freund[3]), les nouveaux journaux sont initialisés par des zéros avant d'être utilisés. Recyclage et initialisation peuvent être désactivés en passant les paramètres `wal_recycle` et `wal_init_zero` à `off`, ce qui n'est conseillé que sur certains systèmes de fichiers comme ZFS ou btrfs.

Dans le cadre d'un archivage PITR et/ou d'une réplication par *log shipping*, le sous-répertoire `pg_wal/archive_status` indique l'état des journaux dans le contexte de l'archivage. Les fichiers `.ready` indiquent les journaux restant à archiver (normalement peu nombreux), les `.done` ceux déjà archivés. Il est possible de connaître facilement le contenu de ce répertoire en utilisant la fonction `pg_ls_archive_statusdir()` :

```
SELECT * FROM pg_ls_archive_statusdir() ORDER BY 1 ;
```

```
             name               | size |      modification
--------------------------------+------+------------------------
 000000010000000000000067.done  |    0 | 2020-09-02 15:52:57+02
 000000010000000000000068.done  |    0 | 2020-09-02 15:52:57+02
 000000010000000000000069.done  |    0 | 2020-09-02 15:52:58+02
 00000001000000000000006A.ready |    0 | 2020-09-02 15:53:53+02
 00000001000000000000006B.ready |    0 | 2020-09-02 15:53:53+02
 00000001000000000000006C.ready |    0 | 2020-09-02 15:53:54+02
 00000001000000000000006D.ready |    0 | 2020-09-02 15:53:54+02
 00000001000000000000006E.ready |    0 | 2020-09-02 15:53:54+02
 00000001000000000000006F.ready |    0 | 2020-09-02 15:53:54+02
 000000010000000000000070.ready |    0 | 2020-09-02 15:53:55+02
 000000010000000000000071.ready |    0 | 2020-09-02 15:53:55+02
```

Le répertoire `pg_xact` contient l'état de toutes les transactions passées ou présentes sur la base (validées, annulées, en sous-transaction ou en cours), comme nous le détaillerons dans le module « Mécanique du moteur transactionnel ».

[3] https://www.postgresql.org/message-id/4xaez4hh3cwkclr77tnkdba6wisueu32vza44iugozai5hklyb%405epf6zpo66aa

Les fichiers contenus dans le répertoire `pg_xact` ne doivent **jamais** être effacés. Ils sont cruciaux au bon fonctionnement de la base.

D'autres répertoires contiennent des fichiers essentiels à la gestion des transactions :

- `pg_commit_ts` contient l'horodatage de la validation de chaque transaction ;
- `pg_multixact` est utilisé dans l'implémentation des verrous partagés (`SELECT … FOR SHARE`) ;
- `pg_serial` est utilisé dans l'implémentation de SSI (`Serializable Snapshot Isolation`) ;
- `pg_snapshots` est utilisé pour stocker les snapshots exportés de transactions ;
- `pg_subtrans` stocke l'imbrication des transactions lors de sous-transactions (les `SAVEPOINTS`) ;
- `pg_twophase` est utilisé pour l'implémentation du *Two-Phase Commit*, aussi appelé `transaction préparée`, `2PC`, ou transaction `XA` dans le monde Java par exemple.

La version 10 a été l'occasion du changement de nom de quelques répertoires pour des raisons de cohérence et pour réduire les risques de fausses manipulations. Jusqu'en 9.6, `pg_wal` s'appelait `pg_xlog`, `pg_xact` s'appelait `pg_clog`.
Les fonctions et outils ont été renommés en conséquence :

- dans les noms de fonctions et d'outils, `xlog` a été remplacé par `wal` (par exemple `pg_switch_xlog` est devenue `pg_switch_wal`) ;
- toujours dans les fonctions, `location` a été remplacé par `lsn`.

1.6.6 Fichiers liés à la réplication

- `pg_logical/`
- `pg_repslot/`

`pg_logical` contient des informations sur la réplication logique.

`pg_replslot` contient des informations sur les slots de réplications, qui sont un moyen de fiabiliser la réplication physique ou logique.

Sans réplication en place, ces répertoires sont quasi-vides. Là encore, il ne faut pas toucher à leur contenu.

1.6.7 Répertoire des tablespaces

- `pg_tblspc/` : tablespaces
  - si vraiment nécessaires
  - liens symboliques ou points de jonction
  - totalement optionnels

Dans PGDATA, le sous-répertoire `pg_tblspc` contient les *tablespaces*, c'est-à-dire des espaces de stockage.

1.6.7.1 Sous Linux

Sous Linux, ce sont des liens symboliques vers un simple répertoire extérieur à PGDATA. Chaque lien symbolique a comme nom l'OID du tablespace (table système `pg_tablespace`). PostgreSQL y crée un répertoire lié aux versions de PostgreSQL et du catalogue, et y place les fichiers de données.

```
postgres=# \db+
                            Liste des tablespaces
    Nom     | Propriétaire |      Emplacement      | … | Taille  |…
------------+--------------+-----------------------+---+---------+-
 froid      | postgres     | /mnt/hdd/pg           |   | 3576 kB |
 pg_default | postgres     |                       |   | 6536 MB |
 pg_global  | postgres     |                       |   | 587 kB  |
```

```
sudo ls -R /mnt/hdd/pg
```

```
/mnt/hdd/pg:
PG_15_202209061

/mnt/hdd/pg/PG_15_202209061:
5

/mnt/hdd/pg/PG_15_202209061/5:
142532  142532_fsm  142532_vm
```

1.6.7.2 Sous Windows

Sous Windows, les liens sont à proprement parler des *Reparse Points* (ou *Junction Points*) :

```
postgres=# \db
          Liste des tablespaces
    Nom     | Propriétaire | Emplacement
------------+--------------+-------------
```

```
 pg_default | postgres     |
 pg_global  | postgres     |
 tbl1       | postgres     | T:\TBL1
```

```
PS P:\PGDATA13> dir 'pg_tblspc/*' | ?{$_.LinkType} | select FullName,LinkType,Target

FullName                    LinkType Target
--------                    -------- ------
P:\PGDATA13\pg_tblspc\105921 Junction {T:\TBL1}
```

Par défaut, `pg_tblspc/` est vide. N'existent alors que les tablespaces `pg_global` (sous-répertoire `global/` des objets globaux à l'instance) et `pg_default` (soit `base/`).

La création d'autres tablespaces est totalement optionnelle.

Leur utilité et leur gestion seront abordés plus loin.

1.6.8 Fichiers des statistiques d'activité

Statistiques d'activité :

- `stats collector` (≤v14) & extensions
- `pg_stat_tmp/` : temporaires
- `pg_stat/` : définitif

`pg_stat_tmp` est, jusqu'en version 15, le répertoire par défaut de stockage des statistiques d'activité de PostgreSQL, comme les entrées-sorties ou les opérations de modifications sur les tables. Ces fichiers pouvant générer une grande quantité d'entrées-sorties, l'emplacement du répertoire peut être modifié avec le paramètre `stats_temp_directory`. Par exemple, Debian place ce paramètre par défaut en `tmpfs` :

```
-- jusque v14
SHOW stats_temp_directory;
```

```
           stats_temp_directory
----------------------------------------
 /var/run/postgresql/14-main.pg_stat_tmp
```

À l'arrêt, les fichiers sont copiés dans le répertoire `pg_stat/`.

PostgreSQL gérant ces statistiques en mémoire partagée à partir de la version 15, le collecteur n'existe plus. Mais les deux répertoires sont encore utilisés par des extensions comme `pg_stat_statements` ou `pg_stat_kcache`.

1.6.9 Autres répertoires

- `pg_dynshmem/`
- `pg_notify/`

`pg_dynshmem` est utilisé par les extensions utilisant de la mémoire partagée dynamique.

`pg_notify` est utilisé par le mécanisme de gestion de notification de PostgreSQL (`LISTEN` et `NOTIFY`) qui permet de passer des messages de notification entre sessions.

1.6.10 Les fichiers de traces (journaux)

- Fichiers texte traçant l'activité
- Très paramétrables
- Gestion des fichiers soit :
  - par PostgreSQL
  - délégués au système d'exploitation (*syslog*, *eventlog*)

Le paramétrage des journaux est très fin. Leur configuration est le sujet est évoquée dans notre première formation[4].

Si `logging_collector` est activé, c'est-à-dire que PostgreSQL collecte lui-même ses traces, l'emplacement de ces journaux se paramètre grâce aux paramètres `log_directory`, le répertoire où les stocker, et `log_filename`, le nom de fichier à utiliser, ce nom pouvant utiliser des échappements comme `%d` pour le jour de la date, par exemple. Les droits attribués au fichier sont précisés par le paramètre `log_file_mode`.

Un exemple pour `log_filename` avec date et heure serait :

```
log_filename = 'postgresql-%Y-%m-%d_%H%M%S.log'
```

La liste des échappements pour le paramètre `log_filename` est disponible dans la page de manuel de la fonction `strftime` sur la plupart des plateformes de type UNIX.

[4] https://dali.bo/h1_html

1.7 RÉSUMÉ

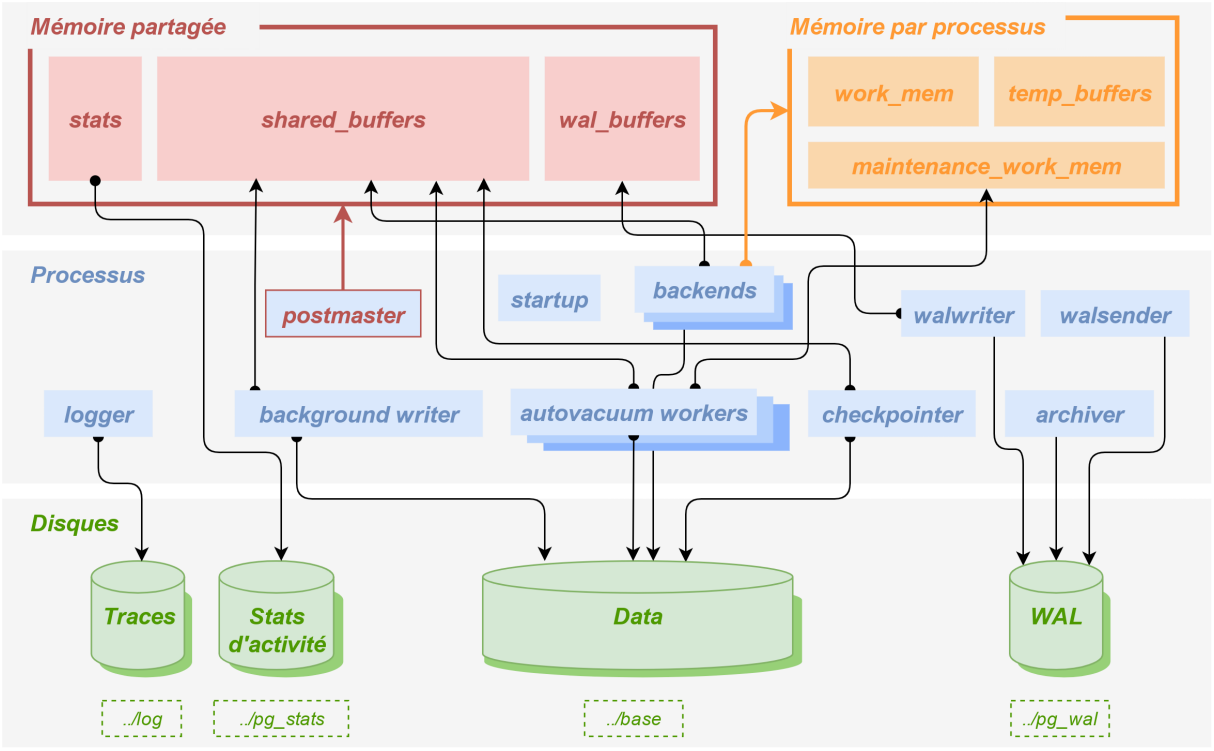


Figure 1/ .1: Architecture de PostgreSQL

Ce schéma ne soit à présent plus avoir aucun secret pour vous.

1.8 CONCLUSION

- PostgreSQL est complexe, avec de nombreux composants
- Une bonne compréhension de cette architecture est la clé d'une bonne administration.
- Pour aller (beaucoup) plus loin :

1.8.1 Questions

N'hésitez pas, c'est le moment !

1.9 QUIZ

https://dali.bo/m1_quiz

1.10 INSTALLATION DE POSTGRESQL DEPUIS LES PAQUETS COMMUNAUTAIRES

L'installation est détaillée ici pour Rocky Linux 8 et 9 (similaire à Red Hat et à d'autres variantes comem Oracle Linux et Fedora), et Debian/Ubuntu.

Elle ne dure que quelques minutes.

1.10.1 Sur Rocky Linux 8 ou 9

ATTENTION : Red Hat, CentOS, Rocky Linux fournissent souvent 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 fortement conseillée.

Installation du dépôt communautaire :

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

- la version majeure de PostgreSQL (ici la 17) ;
- la distribution (ici Rocky Linux 8) ;
- l'architecture (ici x86_64, la plus courante).

Les commandes sont à lancer sous **root** :

```
# 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 17 (client, serveur, librairies, extensions) :

```
# dnf install -y postgresql17-server postgresql17-contrib
```

Les outils clients et les librairies nécessaires seront automatiquement installés.

Une fonctionnalité avancée optionnelle, le JIT (*Just In Time compilation*), nécessite un paquet séparé.

```
# dnf install postgresql17-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-17/bin/postgresql-17-setup initdb
# cat /var/lib/pgsql/17/initdb.log
```

Ce dernier fichier permet de vérifier que tout s'est bien passé et doit finir par :

```
Success. You can now start the database server using:

    /usr/pgsql-17/bin/pg_ctl -D /var/lib/pgsql/17/data/ -l logfile start
```

Chemins :

Objet	Chemin
Binaires	`/usr/pgsql-17/bin`
Répertoire de l'utilisateur **postgres**	`/var/lib/pgsql`
`PGDATA` par défaut	`/var/lib/pgsql/17/data`
Fichiers de configuration	dans `PGDATA/`
Traces	dans `PGDATA/log`

Configuration :

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

Commandes d'administration habituelles :

Démarrage, arrêt, statut, rechargement à chaud de la configuration, redémarrage :

```
# systemctl start postgresql-17
# systemctl stop postgresql-17
# systemctl status postgresql-17
# systemctl reload postgresql-17
# systemctl restart postgresql-17
```

Test rapide de bon fonctionnement et connexion à psql :

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

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

```
# systemctl enable postgresql-17
```

Ouverture du *firewall* pour le port 5432 :

Voir si le *firewall* est actif :

```
# systemctl status firewalld
```

Si c'est le cas, autoriser un accès extérieur :

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

(Rappelons que `listen_addresses` doit être également modifié dans `postgresql.conf`.)

Création d'autres instances :

Si des instances de *versions majeures différentes* doivent être installées, il faut d'abord installer les binaires pour chacune (adapter le numéro dans `dnf install …`) et appeler le script d'installation de chaque version. l'instance par défaut de chaque version vivra dans un sous-répertoire numéroté de `/var/lib/pgsql` automatiquement créé à l'installation. Il faudra juste modifier les ports dans les `postgresql.conf` pour que les instances puissent tourner simultanément.

Si plusieurs instances d'une *même version majeure* (forcément de la même version mineure) doivent cohabiter sur le même serveur, il faut 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 nouveau sous-répertoire de `/var/lib/pgsqsl/17/` (ou l'équivalent pour d'autres versions majeures).

Pour créer une seconde instance, nommée par exemple **infocentre** :

- Création du fichier service de la deuxième instance :

```
# cp /lib/systemd/system/postgresql-17.service \
        /etc/systemd/system/postgresql-17-infocentre.service
```

- Modification de ce dernier fichier avec le nouveau chemin :

```
Environment=PGDATA=/var/lib/pgsql/17/infocentre
```

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

```
# export PGSETUP_INITDB_OPTIONS='--data-checksums --lc-messages=C'
# /usr/pgsql-17/bin/postgresql-17-setup initdb postgresql-17-infocentre
```

- Option 2 : restauration d'une sauvegarde : la procédure dépend de votre outil.
- Adaptation de `/var/lib/pgsql/17/infocentre/postgresql.conf` (port surtout).
- Commandes de maintenance de cette instance :

```
# systemctl [start|stop|reload|status] postgresql-17-infocentre
# systemctl [enable|disable] postgresql-17-infocentre
```

- Ouvrir le nouveau port dans le firewall au besoin.

1.10.2 Sur Debian / Ubuntu

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

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

Installation du dépôt communautaire :

L'installation des dépôts du PGDG est prévue dans le paquet Debian :

```
# apt update
# apt install -y  gnupg2  postgresql-common
# /usr/share/postgresql-common/pgdg/apt.postgresql.org.sh
```

Ce dernier ordre créera le fichier du dépôt `/etc/apt/sources.list.d/pgdg.list` adapté à la distribution en place.

Installation de PostgreSQL 17 :

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

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-17 postgresql-client-17
```

La première instance est automatiquement créée, démarrée et déclarée comme service à lancer au démarrage du système. Elle porte un nom (par défaut `main`).

Elle est immédiatement accessible par l'utilisateur système **postgres**.

Chemins :

Objet	Chemin
Binaires	`/usr/lib/postgresql/17/bin/`
Répertoire de l'utilisateur **postgres**	`/var/lib/postgresql`
PGDATA de l'instance par défaut	`/var/lib/postgresql/17/main`
Fichiers de configuration	dans `/etc/postgresql/17/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, qui demandent en paramètre la version et le nom de l'instance :

```
# pg_ctlcluster 17 main [start|stop|reload|status|restart]
```

Démarrage de l'instance avec le serveur :

C'est en place par défaut, et modifiable dans `/etc/postgresql/17/main/start.conf`.

Ouverture du firewall :

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

Statut des instances du serveur :

```
# pg_lsclusters
```

Test rapide de bon fonctionnement et connexion à psql :

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

Destruction d'une instance :

```
# pg_dropcluster 17 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 :

- optionnellement, `/etc/postgresql-common/createcluster.conf` permet de mettre en place tout d'entrée les *checksums*, les messages en anglais, le format des traces ou un emplacement séparé pour les journaux :

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

- créer une instance :

```
# pg_createcluster 17 infocentre
```

Il est également possible de préciser certains paramètres du fichier `postgresql.conf`, voire les chemins des fichiers (il est conseillé de conserver les chemins par défaut) :

```
# pg_createcluster 17 infocentre \
  --port=12345 \
  --datadir=/PGDATA/17/infocentre \
  --pgoption shared_buffers='8GB' --pgoption work_mem='50MB' \
  --  --data-checksums --waldir=/ssd/postgresql/17/infocentre/journaux
```

- adapter au besoin `/etc/postgresql/17/infocentre/postgresql.conf` ;
- démarrage :

```
# pg_ctlcluster 17 infocentre start
```

1.10.3 Accès à l'instance depuis le serveur même (toutes distributions)

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 (17.0)
Type "help" for help.
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 (17.0)
Type "help" for help.
postgres=#
```

Une authentification par mot de passe est plus sécurisée :

- dans `pg_hba.conf`, paramétrer une authentification par mot de passe pour les accès depuis `localhost` (déjà en place sous Debian) :

```
# IPv4 local connections:
host    all             all             127.0.0.1/32            scram-sha-256
# IPv6 local connections:
host    all             all             ::1/128                 scram-sha-256
```

(Ne pas oublier de recharger la configuration en cas de modification.)

- ajouter un mot de passe à l'utilisateur `postgres` de l'instance :

```
dalibo:~$ sudo -iu postgres psql
psql (17.0)
Type "help" for help.
postgres=# \password
Enter new password for user "postgres":
Enter it again:
postgres=# quit

dalibo:~$ psql -h localhost -U postgres
Password for user postgres:
psql (17.0)
Type "help" for help.
postgres=#
```

- Pour se connecter sans taper le mot de passe à une instance, 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/17/data/log` ou `/var/log/postgresql/`) ;
- toute modification de `pg_hba.conf` ou `postgresql.conf` impliquant de recharger la configuration peut être réalisée par une de ces trois méthodes en fonction du système :

```
root:~# systemctl reload postgresql-17

root:~# pg_ctlcluster 17 main reload

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

1.11 TRAVAUX PRATIQUES

La version en ligne des solutions de ces TP est disponible sur https://dali.bo/m1_solutions.

1.11.1 Processus

But : voir quelques processus de PostgreSQL

Si ce n'est pas déjà fait, démarrer l'instance PostgreSQL.

Lister les processus du serveur PostgreSQL. Qu'observe-t-on ?

Se connecter à l'instance PostgreSQL.

Dans un autre terminal lister de nouveau les processus du serveur PostgreSQL. Qu'observe-t-on ?

Créer une nouvelle base de données nommée **b0**.

Se connecter à la base de données **b0** et créer une table `t1` avec une colonne `id` de type `integer`.

Insérer 10 millions de lignes dans la table `t1` avec :

```
INSERT INTO t1 SELECT generate_series(1, 10000000) ;
```

Dans un autre terminal lister de nouveau les processus du serveur PostgreSQL. Qu'observe-t-on ?

Configurer la valeur du paramètre `max_connections` à `15`.

Redémarrer l'instance PostgreSQL.

Vérifier que la modification de la valeur du paramètre `max_connections` a été prise en compte.

Se connecter 15 fois à l'instance PostgreSQL sans fermer les sessions, par exemple en lançant plusieurs fois :

```
psql -c 'SELECT pg_sleep(1000)' &
```

Se connecter une seizième fois à l'instance PostgreSQL. Qu'observe-t-on ?

Configurer la valeur du paramètre `max_connections` à sa valeur initiale.

1.11.2 Fichiers

But : voir les fichiers de PostgreSQL

Aller dans le répertoire des données de l'instance PostgreSQL. Lister les fichiers.

Aller dans le répertoire `base`. Lister les fichiers.

À quelle base est lié chaque répertoire présent dans le répertoire `base` ? (Voir l'oid de la base dans `pg_database` ou l'utilitaire en ligne de commande `oid2name`)

Créer une nouvelle base de données nommée **b1**. Qu'observe-t-on dans le répertoire `base` ?

Se connecter à la base de données **b1**. Créer une table `t1` avec une colonne `id` de type `integer`.

Récupérer le chemin vers le fichier correspondant à la table `t1` (il existe une fonction `pg_relation_filepath`).

Regarder la taille du fichier correspondant à la table `t1`. Pourquoi est-il vide ?

Insérer une ligne dans la table `t1`. Quelle taille fait le fichier de la table `t1` ?

> Insérer 500 lignes dans la table `t1` avec `generate_series`. Quelle taille fait le fichier de la table `t1` ?

> Pourquoi cette taille pour simplement 501 entiers de 4 octets chacun ?

1.12 TRAVAUX PRATIQUES (SOLUTIONS)

1.12.1 Processus

> Si ce n'est pas déjà fait, démarrer l'instance PostgreSQL.

Sous Rocky Linux, CentOS ou Red Hat en tant qu'utilisateur **root** :

```
# systemctl start postgresql-15
```

> Lister les processus du serveur PostgreSQL. Qu'observe-t-on ?

En tant qu'utilisateur **postgres** :

```
$ ps -o pid,cmd fx
    PID CMD
  27886 -bash
  28666  \_ ps -o pid,cmd fx
  27814 /usr/pgsql-17/bin/postmaster -D /var/lib/pgsql/15/data/
  27815  \_ postgres: logger
  27816  \_ postgres: checkpointer
  27817  \_ postgres: background writer
  27819  \_ postgres: walwriter
  27820  \_ postgres: autovacuum launcher
  27821  \_ postgres: logical replication launcher
```

> Se connecter à l'instance PostgreSQL.

```
$ psql postgres
psql (15.0)
Type "help" for help.

postgres=#
```

> Dans un autre terminal lister de nouveau les processus du serveur PostgreSQL. Qu'observe-t-on ?

```
$ ps -o pid,cmd fx
    PID CMD
  28746 -bash
  28779  \_ psql
  27886 -bash
  28781  \_ ps -o pid,cmd fx
  27814 /usr/pgsql-17/bin/postmaster -D /var/lib/pgsql/15/data/
  27815  \_ postgres: logger
  27816  \_ postgres: checkpointer
  27817  \_ postgres: background writer
  27819  \_ postgres: walwriter
  27820  \_ postgres: autovacuum launcher
  27821  \_ postgres: logical replication launcher
  28780  \_ postgres: postgres postgres [local] idle
```

Il y a un nouveau processus (ici PID 28780) qui va gérer l'exécution des requêtes du client psql.

> Créer une nouvelle base de données nommée **b0**.

Depuis le shell, en tant que **postgres** :

```
$ createdb b0
```

Alternativement, depuis la session déjà ouverte dans `psql` :

```
CREATE DATABASE b0;
```

> Se connecter à la base de données **b0** et créer une table `t1` avec une colonne `id` de type `integer`.

Pour se connecter depuis le shell :

```
psql b0
```

ou depuis la session `psql` :

```
\c b0
```

Création de la table :

```
CREATE TABLE t1 (id integer);
```

> Insérer 10 millions de lignes dans la table `t1` avec :
>
> ```
> INSERT INTO t1 SELECT generate_series(1, 10000000) ;
> ```

```
INSERT INTO t1 SELECT generate_series(1, 10000000);
```

```
INSERT 0 10000000
```

> Dans un autre terminal lister de nouveau les processus du serveur PostgreSQL. Qu'observe-t-on ?

```
$ ps -o pid,cmd fx
    PID CMD
  28746 -bash
  28779  \_ psql
  27886 -bash
  28781  \_ ps -o pid,cmd fx
  27814 /usr/pgsql-17/bin/postmaster -D /var/lib/pgsql/15/data/
  27815  \_ postgres: logger
  27816  \_ postgres: checkpointer
  27817  \_ postgres: background writer
  27819  \_ postgres: walwriter
  27820  \_ postgres: autovacuum launcher
  27821  \_ postgres: logical replication launcher
  28780  \_ postgres: postgres postgres [local] INSERT
```

Le processus serveur exécute l'`INSERT`, ce qui se voit au niveau du nom du processus. Seul est affiché le dernier ordre SQL (*ie* le mot `INSERT` et non pas la requête complète).

> Configurer la valeur du paramètre `max_connections` à `15`.

La première méthode est d'ouvrir le fichier de configuration `postgresql.conf` et de modifier la valeur du paramètre `max_connections` :

```
max_connections = 15
```

La seconde méthode se fait directement depuis psql en tant que superutilisateur :

```
ALTER SYSTEM SET max_connections TO 15 ;
```

Ce dernier ordre fera apparaître une ligne dans le fichier `/var/lib/pgsql/15/data/postgresql.auto.conf`.

Dans les deux cas, la prise en compte n'est pas automatique. Pour ce paramètre, il faut redémarrer l'instance PostgreSQL.

> Redémarrer l'instance PostgreSQL.

En tant qu'utilisateur **root** :

```
# systemctl restart postgresql-15
```

> Vérifier que la modification de la valeur du paramètre `max_connections` a été prise en compte.

```
postgres=# SHOW max_connections ;

 max_connections
-----------------
 15
```

> Se connecter 15 fois à l'instance PostgreSQL sans fermer les sessions, par exemple en lançant plusieurs fois :
>
> ```
> psql -c 'SELECT pg_sleep(1000)' &
> ```

Il est possible de le faire manuellement ou de l'automatiser avec ce petit script shell :

```
$ for i in $(seq 1 15); do psql -c "SELECT pg_sleep(1000);" postgres & done
[1] 998
[2] 999
...
[15] 1012
```

> Se connecter une seizième fois à l'instance PostgreSQL. Qu'observe-t-on ?

```
$ psql postgres
psql: FATAL:  sorry, too many clients already
```

Il est impossible de se connecter une fois que le nombre de connexions a atteint sa limite configurée avec `max_connections`. Il faut donc attendre que les utilisateurs se déconnectent pour accéder de nouveau au serveur.

Configurer la valeur du paramètre `max_connections` à sa valeur initiale.

Si le fichier de configuration `postgresql.conf` avait été modifié, restaurer la valeur du paramètre `max_connections` à 100.

Si `ALTER SYSTEM` a été utilisée, il faudrait pouvoir entrer dans psql :

```
ALTER SYSTEM RESET max_connections ;
```

Mais cela ne fonctionne pas si toutes les connexions réservées à l'utilisateur ont été consommées (paramètre `superuser_reserved_connections`, par défaut à 3). Dans ce cas, il n'y a plus qu'à aller dans le fichier de configuration `/var/lib/pgsql/15/data/postgresql.auto.conf`, pour supprimer la ligne avec le paramètre `max_connections` avant de redémarrer l'instance PostgreSQL.

Il est déconseillé de modifier `postgresql.auto.conf` à la main, mais pour le TP, nous nous permettons quelques libertés.

Toutefois si l'instance est démarrée et qu'il est encore possible de s'y connecter, le plus propre est ceci :

```
ALTER SYSTEM RESET max_connections ;
```

Puis redémarrer PostgreSQL : toutes les connexions en cours vont être coupées.

```
# systemctl restart postgresql-15
FATAL:  terminating connection due to administrator command
server closed the connection unexpectedly
    This probably means the server terminated abnormally
    before or while processing the request.
[...]
FATAL:  terminating connection due to administrator command
server closed the connection unexpectedly
    This probably means the server terminated abnormally
    before or while processing the request.
connection to server was lost
```

Il est à présent possible de se reconnecter. Vérifier que cela a été pris en compte :

```
postgres=# SHOW max_connections ;

 max_connections
-----------------
 100
```

1.12.2 Fichiers

Aller dans le répertoire des données de l'instance PostgreSQL. Lister les fichiers.

En tant qu'utilisateur système **postgres** :

```
 echo $PGDATA
/var/lib/pgsql/15/data

$ cd $PGDATA

$ ls -al
total 68
drwx------. 7 postgres postgres    59 Nov  4 09:55 base
-rw-------. 1 postgres postgres    30 Nov  4 10:38 current_logfiles
drwx------. 2 postgres postgres  4096 Nov  4 10:38 global
drwx------. 2 postgres postgres    58 Nov  4 07:58 log
drwx------. 2 postgres postgres     6 Nov  3 14:11 pg_commit_ts
drwx------. 2 postgres postgres     6 Nov  3 14:11 pg_dynshmem
-rw-------. 1 postgres postgres  4658 Nov  4 09:50 pg_hba.conf
-rw-------. 1 postgres postgres  1685 Nov  3 14:16 pg_ident.conf
drwx------. 4 postgres postgres    68 Nov  4 10:38 pg_logical
drwx------. 4 postgres postgres    36 Nov  3 14:11 pg_multixact
drwx------. 2 postgres postgres     6 Nov  3 14:11 pg_notify
drwx------. 2 postgres postgres     6 Nov  3 14:11 pg_replslot
drwx------. 2 postgres postgres     6 Nov  3 14:11 pg_serial
drwx------. 2 postgres postgres     6 Nov  3 14:11 pg_snapshots
drwx------. 2 postgres postgres     6 Nov  4 10:38 pg_stat
drwx------. 2 postgres postgres    35 Nov  4 10:38 pg_stat_tmp
drwx------. 2 postgres postgres    18 Nov  3 14:11 pg_subtrans
drwx------. 2 postgres postgres     6 Nov  3 14:11 pg_tblspc
drwx------. 2 postgres postgres     6 Nov  3 14:11 pg_twophase
-rw-------. 1 postgres postgres     3 Nov  3 14:11 PG_VERSION
drwx------. 3 postgres postgres    92 Nov  4 09:55 pg_wal
drwx------. 2 postgres postgres    18 Nov  3 14:11 pg_xact
-rw-------. 1 postgres postgres    88 Nov  3 14:11 postgresql.auto.conf
-rw-------. 1 postgres postgres 29475 Nov  4 09:36 postgresql.conf
-rw-------. 1 postgres postgres    58 Nov  4 10:38 postmaster.opts
-rw-------. 1 postgres postgres   104 Nov  4 10:38 postmaster.pid
```

Aller dans le répertoire `base`. Lister les fichiers.

```
$ cd base

$ ls -al
total 60
drwx------  8 postgres postgres   78 Nov  4 16:21 .
drwx------ 20 postgres postgres 4096 Nov  4 15:33 ..
drwx------. 2 postgres postgres 8192 Nov  4 10:38 1
drwx------. 2 postgres postgres 8192 Nov  4 09:50 16404
drwx------. 2 postgres postgres 8192 Nov  3 14:11 4
drwx------. 2 postgres postgres 8192 Nov  4 10:39 5
drwx------  2 postgres postgres    6 Nov  3 15:58 pgsql_tmp
```

> À quelle base est lié chaque répertoire présent dans le répertoire `base` ? (Voir l'oid de la base dans `pg_database` ou l'utilitaire en ligne de commande `oid2name`)

Chaque répertoire correspond à une base de données. Le numéro indiqué est un identifiant système (OID). Il existe deux moyens pour récupérer cette information :

- directement dans le catalogue système `pg_database` :

```
$ psql postgres
```

```
psql (15.0)
Type "help" for help.
```

```
postgres=# SELECT oid, datname FROM pg_database ORDER BY oid::text;
```

```
  oid  |  datname
-------+-----------
     1 | template1
 16404 | b0
     4 | template0
     5 | postgres
```

- avec l'outil `oid2name` (à installer au besoin via le paquet `postgresql15-contrib`) :

```
$ /usr/pgsql-17/bin/oid2name
All databases:
    Oid  Database Name  Tablespace
----------------------------------
  16404            b0  pg_default
      5      postgres  pg_default
      4     template0  pg_default
      1     template1  pg_default
```

Donc ici, le répertoire `1` correspond à la base `template1`, et le répertoire `5` à la base `postgres` (ces nombres peuvent changer suivant l'installation).

> Créer une nouvelle base de données nommée **b1**. Qu'observe-t-on dans le répertoire `base` ?

```
$ createdb b1
```

```
$ ls -al
total 60
drwx------  8 postgres postgres   78 Nov  4 16:21 .
drwx------ 20 postgres postgres 4096 Nov  4 15:33 ..
drwx------. 2 postgres postgres 8192 Nov  4 10:38 1
drwx------. 2 postgres postgres 8192 Nov  4 09:50 16404
drwx------. 2 postgres postgres 8192 Nov  4 09:55 16405
drwx------. 2 postgres postgres 8192 Nov  3 14:11 4
drwx------. 2 postgres postgres 8192 Nov  4 10:39 5
drwx------  2 postgres postgres    6 Nov  3 15:58 pgsql_tmp
```

Un nouveau sous-répertoire est apparu, nommé `16405`. Il correspond bien à la base `b1` d'après `oid2name`.

> Se connecter à la base de données **b1**. Créer une table `t1` avec une colonne `id` de type `integer`.

```
$ psql b1
psql (15.0)
Type "help" for help.

b1=# CREATE TABLE t1(id integer);

CREATE TABLE
```

> Récupérer le chemin vers le fichier correspondant à la table `t1` (il existe une fonction `pg_relation_filepath`).

La fonction a pour définition :

```
b1=# \df pg_relation_filepath
                                 List of functions
   Schema   |         Name          | Result data type | Argument data types | Type
------------+-----------------------+------------------+---------------------+------
 pg_catalog | pg_relation_filepath | text             | regclass            | func
```

L'argument `regclass` peut être l'OID de la table, ou son nom.

L'emplacement du fichier sur le disque est donc :

```
b1=# SELECT current_setting('data_directory') || '/' || pg_relation_filepath('t1')
     AS chemin;

                chemin
---------------------------------------
 /var/lib/pgsql/15/data/base/16405/16406
```

> Regarder la taille du fichier correspondant à la table `t1`. Pourquoi est-il vide ?

```
$ ls -l /var/lib/pgsql/15/data/base/16393/16398
-rw-------. 1 postgres postgres 0 Nov  4 10:42
 ↪  /var/lib/pgsql/15/data/base/16405/16406
```

La table vient d'être créée. Aucune donnée n'a encore été ajoutée. Les métadonnées se trouvent dans d'autres tables (des catalogues systèmes). Donc il est logique que le fichier soit vide.

> Insérer une ligne dans la table `t1`. Quelle taille fait le fichier de la table `t1` ?

```
b1=# INSERT INTO t1 VALUES (1);
INSERT 0 1

$ ls -l /var/lib/pgsql/15/data/base/16393/16398
-rw-------. 1 postgres postgres 8192 Nov  4 12:40
 ↪  /var/lib/pgsql/15/data/base/16405/16406
```

Il fait à présent 8 ko. En fait, PostgreSQL travaille par blocs de 8 ko. Si une ligne ne peut être placée dans un espace libre d'un bloc existant, un bloc entier est ajouté à la table.

Vous pouvez consulter le fichier avec la commande `hexdump -x <nom du fichier>` (faites un `CHECKPOINT` avant pour être sûr qu'il soit écrit sur le disque).

> Insérer 500 lignes dans la table `t1` avec `generate_series`. Quelle taille fait le fichier de la table `t1` ?

```
b1=# INSERT INTO t1 SELECT generate_series(1, 500);
INSERT 0 500
```

```
$ ls -l /var/lib/pgsql/15/data/base/16393/16398
-rw-------. 1 postgres postgres 24576 Nov  4 12:41
 ↪  /var/lib/pgsql/15/data/base/16405/16406
```

Le fichier fait maintenant 24 ko, soit 3 blocs de 8 ko.

> Pourquoi cette taille pour simplement 501 entiers de 4 octets chacun ?

Nous avons enregistré 501 entiers dans la table. Un entier de type `int4` prend 4 octets. Donc nous avons 2004 octets de données utilisateurs. Et pourtant, nous arrivons à un fichier de 24 ko.

En fait, PostgreSQL enregistre aussi dans chaque bloc des informations systèmes en plus des données utilisateurs. Chaque bloc contient un en-tête, des pointeurs, et l'ensemble des lignes du bloc. Chaque ligne contient les colonnes utilisateurs mais aussi des colonnes système. La requête suivante permet d'en savoir plus sur les colonnes présentes dans la table :

```
b1=# SELECT CASE WHEN attnum<0 THEN 'systeme' ELSE 'utilisateur' END AS type,
     attname, attnum, typname, typlen,
     sum(typlen) OVER (PARTITION BY attnum<0) AS longueur_tot
     FROM pg_attribute a
     JOIN pg_type t ON t.oid=a.atttypid
     WHERE attrelid ='t1'::regclass
     ORDER BY attnum;
```

```
    type     | attname   | attnum | typname | typlen | longueur_tot
-------------+-----------+--------+---------+--------+--------------
 systeme     | tableoid  |     -6 | oid     |      4 |           26
 systeme     | cmax      |     -5 | cid     |      4 |           26
 systeme     | xmax      |     -4 | xid     |      4 |           26
 systeme     | cmin      |     -3 | cid     |      4 |           26
 systeme     | xmin      |     -2 | xid     |      4 |           26
 systeme     | ctid      |     -1 | tid     |      6 |           26
 utilisateur | id        |      1 | int4    |      4 |            4
```

Vous pouvez voir ces colonnes système en les appelant explicitement :

```
SELECT cmin, cmax, xmin, xmax, ctid, *
FROM t1 ;
```

L'en-tête de chaque ligne pèse 26 octets dans le cas général. Dans notre cas très particulier avec une seule petite colonne, c'est très défavorable mais ce n'est généralement pas le cas.

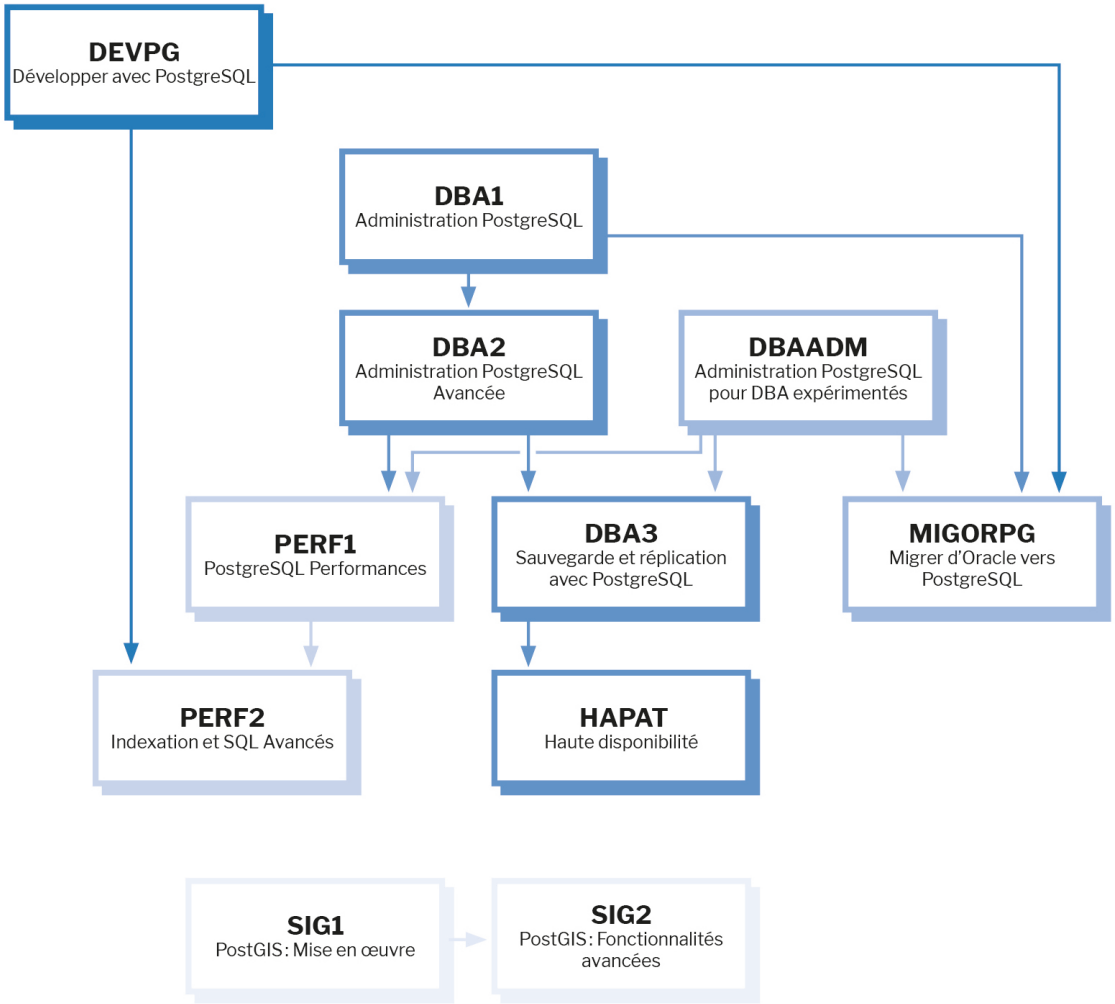
Avec 501 lignes de 26+4 octets, nous obtenons 15 ko. Chaque bloc possède quelques informations de maintenance : nous dépassons alors 16 ko, ce qui explique pourquoi nous en sommes à 24 ko (3 blocs).

Les formations Dalibo

Retrouvez nos formations et le calendrier sur https://dali.bo/formation

Pour toute information ou question, n'hésitez pas à nous écrire sur contact@dalibo.com.

Cursus des formations



Retrouvez nos formations dans leur dernière version :

- 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

Les livres blancs

- Migrer d'Oracle à PostgreSQL
  https://dali.bo/dlb01
- Industrialiser PostgreSQL
  https://dali.bo/dlb02
- Bonnes pratiques de modélisation avec PostgreSQL
  https://dali.bo/dlb04
- Bonnes pratiques de développement avec PostgreSQL
  https://dali.bo/dlb05

Téléchargement gratuit

Les versions électroniques de nos publications sont disponibles gratuitement sous licence open source ou sous licence Creative Commons.