pg_dump ou pas pg_dump, telle est la question

data egret

Your remote PostgreSQL DBA team

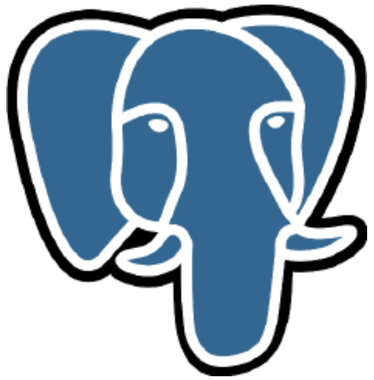
Stefan FERCOT

contact@dataegret.com

Stefan Fercot

- Consultant/Expert PostgreSQL Senior **@Data Egret**
  - Conseil, support et formation PostgreSQL
- Fan & contributeur de pgBackRest
- Contributeur PostgreSQL reconnu
- aka. pgstef
- https://pgstef.github.io

PostgreSQL



https://www.postgresql.org/docs
https://docs.postgresql.fr

pg_dump ou pas pg_dump, telle est la question

Chasseurs de mythes

Objectifs

- Comprendre les types de sauvegardes PostgreSQL
- Savoir quand utiliser `pg_dump` (ou non)
- Acquérir des réflexes de sauvegarde, même en dev/test

Pourquoi avons-nous besoin de sauvegardes ?

- Qu’est-ce qui pourrait mal tourner ?
  - Panne d’infra (stockage, réseau)
  - Erreur humaine
  - Corruption du système ou des données
  - Migration de schéma applicatif problématique
- Comment dupliquer un environnement de production ?
  - Ou créer un environnement CI/CD reflétant la réalité ?

Types de sauvegardes PostgreSQL

Logique vs Physique

Type	Outil	Avantages	Inconvénients
Logique	`pg_dump`	Simple, portable	Lent
Physique	`pg_basebackup`	Rapide, complet, PITR	Moins portable, plus complexe

Copier un fichier .docx ≠ sauvegarder le contenu d'un wiki actif

pg_dump

https://docs.postgresql.fr/current/app-pgdump.html

```
$ pg_dump --help
pg_dump dumps a database as a text file or to other formats.

Usage:
  pg_dump [OPTION]... [DBNAME]
```

Format texte

- Format par défaut
- Fichier de scripts SQL en texte simple

```
$ pg_dump pagila > pagila.sql
$ du -hs pagila.sql
103M  pagila.sql

$ psql -X -d pagila_restored -f pagila.sql
```

Contenu du .sql

```sql
CREATE TABLE public.customer (
    customer_id integer DEFAULT nextval('public.customer_customer_id_seq'::regclass) NOT NULL,
    store_id integer NOT NULL,
    first_name text NOT NULL,
    last_name text NOT NULL,
    email text,
    address_id integer NOT NULL,
    activebool boolean DEFAULT true NOT NULL,
    create_date date DEFAULT CURRENT_DATE NOT NULL,
    last_update timestamp with time zone DEFAULT now(),
    active integer
);
```

```sql
COPY public.customer (customer_id, store_id, first_name, last_name, email, address_id, ...
```

```sql
CREATE INDEX idx_last_name ON public.customer USING btree (last_name);
```

Format “custom”

- Archive personnalisée utilisable par `pg_restore`
- Compressé par défaut

```
$ pg_dump -Fc pagila > pagila.dump
$ du -hs pagila.dump
13M pagila.dump

$ pg_restore -d pagila_restored pagila.dump
```

En résumé

- Export logique = flexibilité
  - Niveau base de données
  - Précision sur les objets: tables, séquences,…
  - La cohérence entre plusieurs bases est un défi
- Restauration
  - Base de données propre (pas de fragmentation)
  - Attention aux `CREATE INDEX` …
    - et à la collection des statistiques (`ANALYSE`) !

Tailles des données ↗ Performance ↘

Fragmentation des données

- Qu’est-ce que c’est ?
  - UPDATE / DELETE laissent des lignes mortes (*dead tuples*)
- Impacts sur les performances
  - Taille des tables augmente -> lectures inutiles
  - Index ralentis -> lignes mortes toujours référencées

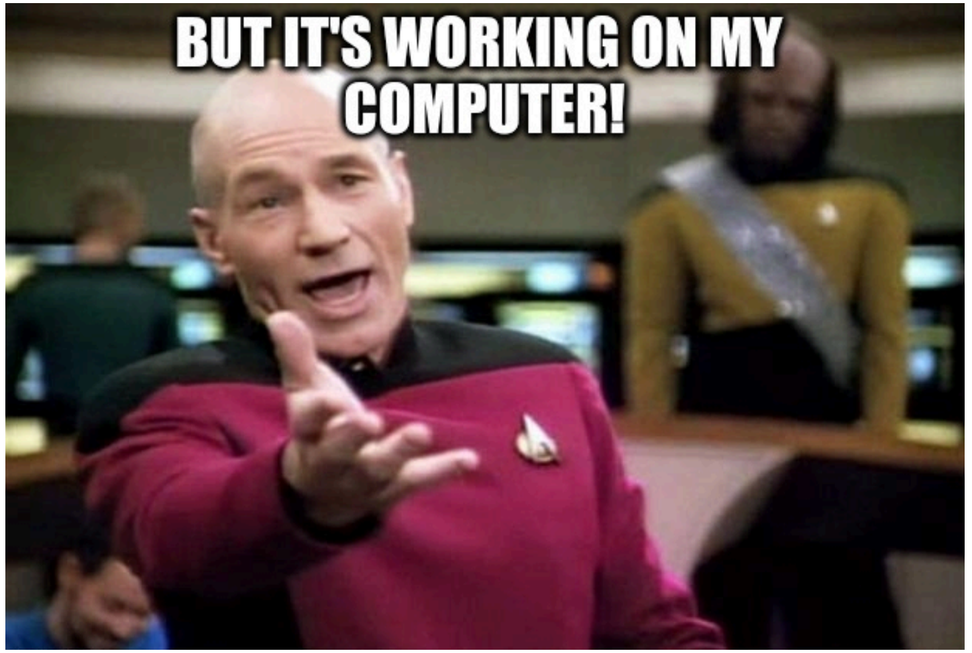
Statistiques

- Utilisées par le planificateur de requêtes
  - pour optimiser les plans d’exécution
- Exemple
  - Distribution des valeurs dans les colonnes
  - Nombre de lignes
  - Fréquence des valeurs nulles ou distinctes
- Impact sur les performances -> mauvais plans de requêtes
  - Statistiques obsolètes ou imprécises
  - Surestimation ou sous-estimation du volume de données

VACUUM / ANALYZE : réflexes essentiels

Commande	Rôle principal	Quand l'utiliser ?
`VACUUM`	Identifie les **lignes mortes**	Après des `DELETE` / `UPDATE` massifs
`ANALYZE`	Met à jour les **statistiques**	Avant des requêtes complexes ou après un gros chargement
`VACUUM FULL`	**Réécrit physiquement** la table	Pour défragmenter
Autovacuum	Automatique, gère `VACUUM` et `ANALYZE`	À **surveiller** et **ajuster** si besoin

Je te jure, en dev ça marchait !



pg_dump, ou pas pg_dump ?

- `pg_dump` n'est pas forcément adapté
  - à de la sauvegarde de forte volumétrie
  - pour une copie "exacte" d'un système de production

Aller plus loin ?

- Sauvegardes physiques
  - Copie des fichiers de la base de données depuis le système de fichiers
  - Cohérence des données grâce aux WAL (journaux de transactions)

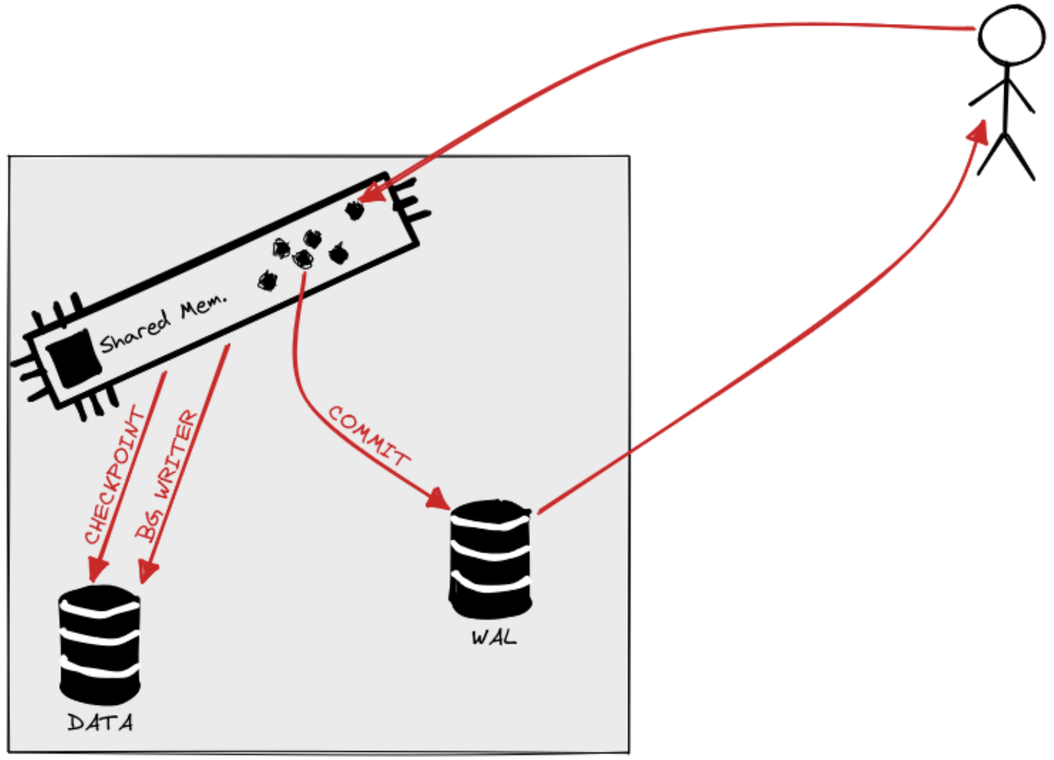
Qu’est-ce que “WAL” ?

- *Write-Ahead Log*
  - Journal des transactions (aussi appelé xlog)
- Taille par défaut des segments : 16 Mo
- Stockés dans le répertoire `pg_wal`
- Conçu pour éviter les pertes de données dans la plupart des situations

Write-Ahead Log (WAL)

- Les transactions sont écrites de manière séquentielle
  - Le COMMIT se fait lorsque les données sont synchronisées sur le disque
- Rejeu du WAL après un crash
  - Permet de rétablir la cohérence de la base de données

Modification des données



Modification des données (2)

- Les transactions modifient les données dans la mémoire partagée (`shared_buffers`)
- Les checkpoints et le processus *background writer*…
  - … écrivent toutes les pages modifiées (“dirty buffers”) sur disque

pg_basebackup

https://docs.postgresql.fr/current/app-pgbasebackup.html

- Prend une copie au niveau du système de fichiers
  - via une connexion de *Streaming Replication*
- Récupère les archives WAL pendant (ou après) la copie

```
$ pg_basebackup --help
pg_basebackup takes a base backup of a running PostgreSQL server.
```

Format `plain`

```
$ pg_basebackup --format=plain --wal-method=stream \
 --checkpoint=fast --progress -D /var/lib/pgsql/17/backups/backup_1

$ du -hs /var/lib/pgsql/17/backups/backup_1
159M  /var/lib/pgsql/17/backups/backup_1

$ cd /var/lib/pgsql/17/backups/backup_1
$ du -hs *
4.0K  backup_label
200K  backup_manifest
142M  base
16M   pg_wal
32K   postgresql.conf
...
```

Format `tar`

```
$ pg_basebackup --format=tar --gzip --wal-method=stream \
 --checkpoint=fast --progress -D /var/lib/pgsql/17/backups/backup_2

$ du -hs /var/lib/pgsql/17/backups/backup_2/*
200K  /var/lib/pgsql/17/backups/backup_2/backup_manifest
21M /var/lib/pgsql/17/backups/backup_2/base.tar.gz
20K /var/lib/pgsql/17/backups/backup_2/pg_wal.tar.gz
```

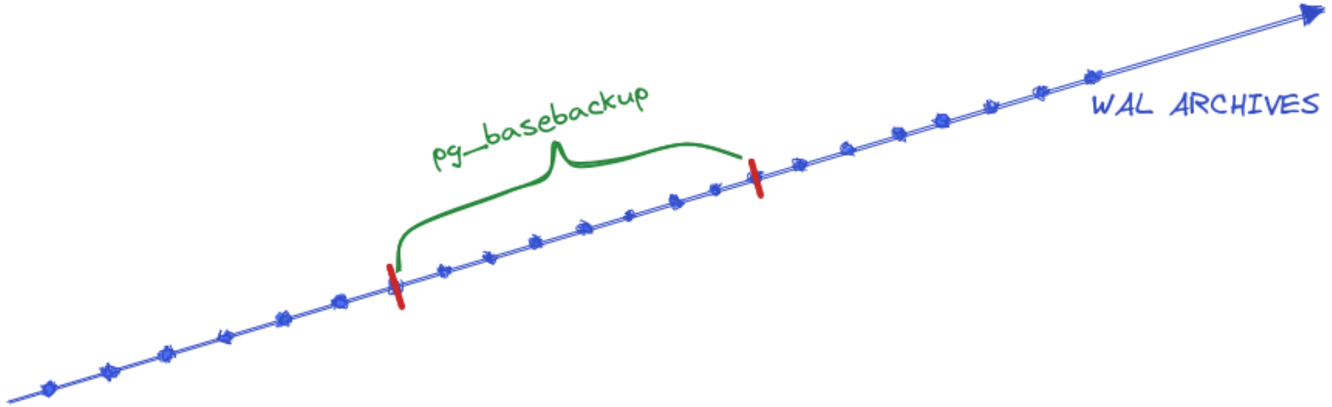
pg_dump, ou pas pg_dump ?

- `pg_basebackup` fait une copie exacte des fichiers de l'instance
- Pas de granularité (instance complète)
- Moins de flexibilité (même OS, version de PostgreSQL)

Point-In-Time Recovery (PITR)

- Restaurer l’instance à un moment précis dans le temps
- Combiner :
  - Une sauvegarde au niveau du système de fichiers
  - L’archivage continu des fichiers WAL
- Pour restaurer :
  - On restaure la sauvegarde système
  - Puis on rejoue les fichiers WAL archivés

Archivage des fichiers WAL



Comment conserver une copie des fichiers WAL?

Mise en place

- 2 possibilités
  - `archive_command`
    - Exemple : `'cp %p /mnt/serveur/repertoire_archive/%f'`
  - `pg_receivewal` (via *Streaming Replication*)

Avantages PITR

- Sauvegarde à chaud
- Moins de pertes de données
- Il n’est pas obligatoire de rejouer les fichiers WAL jusqu’à la fin :
  - Possibilité de s’arrêter à un instant précis (avant une erreur par exemple)

Inconvénients

- Sauvegarde (et restauration) de l'intégralité de l'instance
- Nécessite beaucoup d'espace de stockage (données + archives WAL)
- Le nettoyage des WAL est bloqué en cas d'échec de l'archivage
- Moins simple à utiliser que `pg_dump`

Mise en garde

Ne le faites pas à la main, utilisez des outils de sauvegarde (et de restauration) !

Demo PITR

```
$ createdb pgbench
$ /usr/pgsql-17/bin/pgbench -i -s 600 pgbench
$ /usr/pgsql-17/bin/pgbench -c 4 -j 2 -T 300 pgbench
```

```
archive_mode = on
archive_command = 'test ! -f /backup_space/archives/%f && cp %p /backup_space/archives/%f'
```

Sauvegarde

```
$ pg_basebackup -D "/backup_space/backups/ma-sauvegarde" \
 --format=plain --wal-method=none --checkpoint=fast --progress
NOTICE:  all required WAL segments have been archived
9233844/9233844 kB (100%), 1/1 tablespace
```

Le moment “Oups”…

```
SELECT pg_create_restore_point('voxxedlu-demo');
BEGIN;
    SELECT pg_current_wal_lsn(), current_timestamp;
    DELETE FROM pgbench_tellers;
COMMIT;
BEGIN;
    CREATE TABLE important_table (field text);
    INSERT INTO important_table VALUES ('pg_dump ou pas pg_dump ?');
COMMIT;
SELECT pg_switch_wal();
```

Informations utiles

```
pgbench=# SELECT pg_create_restore_point('voxxedlu-demo');
```

Crée un enregistrement marqueur nommé dans le journal de transactions. Ce marqueur peut ensuite être utilisé comme cible de restauration.

```
recovery_target_name = 'voxxedlu-demo'
```

Informations utiles (2)

```
pgbench=*# SELECT pg_current_wal_lsn(), current_timestamp;
 pg_current_wal_lsn |      current_timestamp
--------------------+-------------------------------
 4/68000060         | 2025-05-28 12:11:03.46935+00
(1 row)
```

`recovery_target_lsn` ou `recovery_target_time`

N’oubliez pas de pratiquer !

Schrödinger’s Law of Backups

> *L’état de toute sauvegarde est inconnu jusqu’à ce qu’une restauration soit tentée.*

État initial

```
pgbench=# SELECT count(*) from pgbench_tellers;
 count
-------
     0
(1 row)

pgbench=# SELECT * FROM important_table;
           field
--------------------------
 pg_dump ou pas pg_dump ?
(1 row)
```

Exemple de restauration

```
$ touch /var/lib/pgsql/17/data/recovery.signal
```

```
# postgresql(.auto).conf
restore_command = 'cp /backup_space/archives/%f %p'
recovery_target_name = 'voxxedlu-demo'
```

```
$ cat /var/lib/pgsql/17/data/backup_label
START WAL LOCATION: 4/67000028 (file 000000010000000400000067)
...
```

Vérifiez les logs

```
LOG:  starting backup recovery with redo LSN 4/67000028, checkpoint LSN 4/67000080, on timeline ID 1
LOG:  restored log file "000000010000000400000067" from archive
LOG:  starting point-in-time recovery to "voxxedlu-demo"
LOG:  redo starts at 4/67000028
LOG:  restored log file "000000010000000400000068" from archive
LOG:  completed backup recovery with redo LSN 4/67000028 and end LSN 4/67000158
LOG:  consistent recovery state reached at 4/67000158
LOG:  database system is ready to accept read-only connections
LOG:  recovery stopping at restore point "voxxedlu-demo", time 2025-05-28 12:11:03.469018+00
LOG:  pausing at the end of recovery
HINT:  Execute pg_wal_replay_resume() to promote.
```

Vérifiez les données

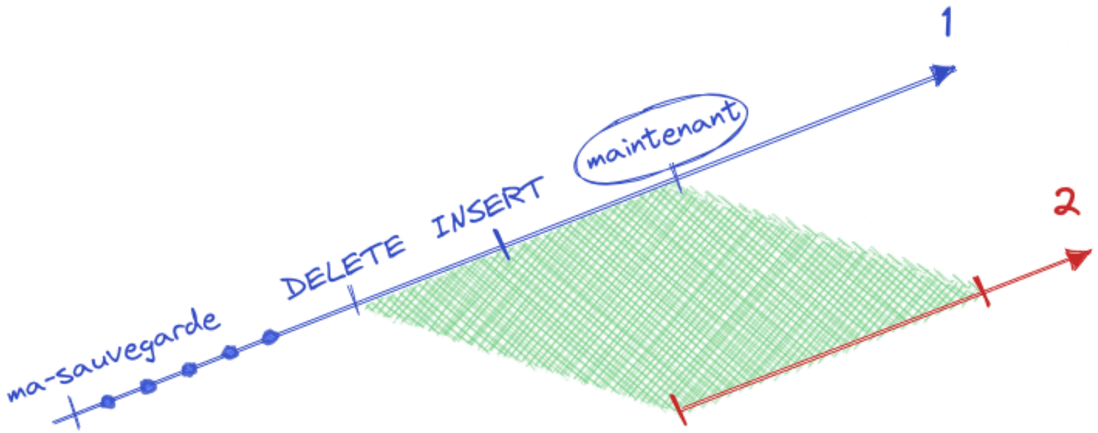
```
pgbench=# SELECT count(*) from pgbench_tellers;
 count
-------
  6000
(1 row)

pgbench=# SELECT * FROM important_table;
ERROR:  relation "important_table" does not exist
LINE 1: SELECT * FROM important_table;
```

Finaliser la restauration

```
psql -c "SELECT pg_wal_replay_resume();"
```

```
LOG:  selected new timeline ID: 2
LOG:  archive recovery complete
LOG:  database system is ready to accept connections
```

Que s’est-il passé ?



Une nouvelle ligne temporelle a été créée !

Et nos archives WAL ?

```
$ ls /backup_space/archives/
000000010000000400000067
000000010000000400000067.00000028.backup
000000010000000400000068
000000020000000400000068
00000002.history

$ cat /backup_space/archives/00000002.history
1 4/680000C8  at restore point "voxxedlu-demo"
```

En résumé

- `pg_dump` est utile, mais pas suffisant
- Comprendre ses limites = ✨ sérénité ✨
- Le vrai backup est celui qu'on peut restaurer !

Message à retenir :

Les sauvegardes ne sont pas qu'un "truc de DBA".
*Ce sont des **outils de développement responsables**.*

data egret

Your remote PostgreSQL DBA team

Vos bases PostgreSQL entre de bonnes mains, vos équipes concentrées sur l'innovation et les nouvelles fonctionnalités.

- **Migration**
- **Audit** PostgreSQL
- **Optimisation** des performances
- Sauvegardes et Haute-Disponibilité
- Revue **d'architecture**
- Conseil pour les équipes **Data Science et Analytics**
- **Formation** pour développeurs

Open Alliance
For PostgreSQL Education

OSB Open Source Business ALLIANCE
Bundesverband für digitale Souveränität e.V.

MEMBER

Merci !

Besoin d'un coup de pouce avec PostgreSQL ?

Contactez Data Egret
pour parler PostgreSQL
et notamment de formations !