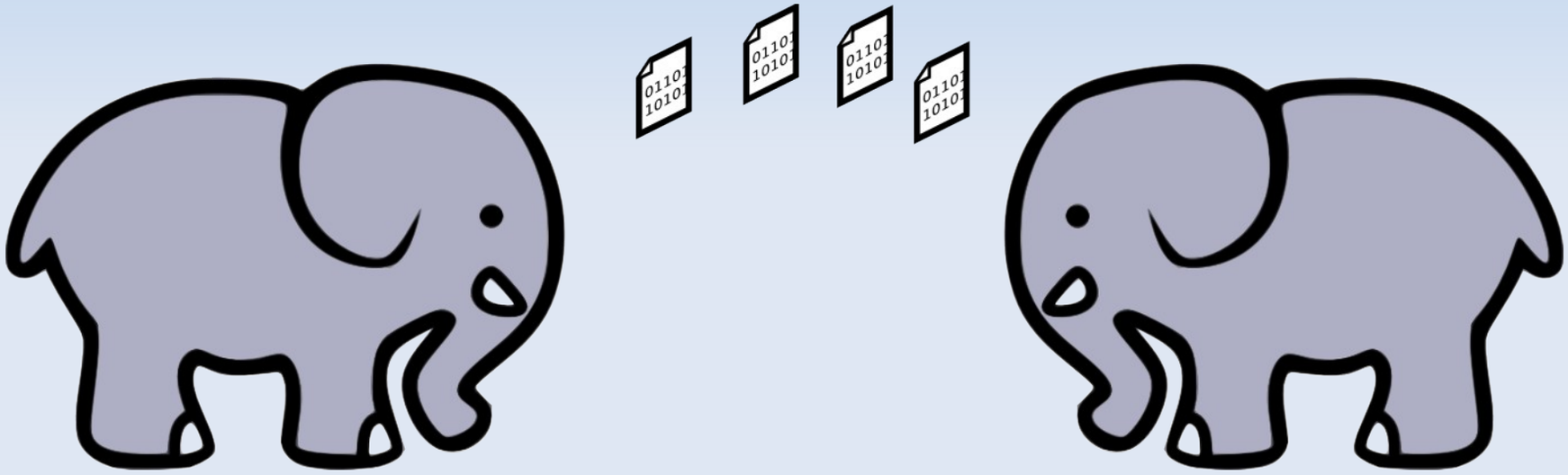
PostgreSQL : PITR et LogShipping

Jean-Christophe Arnu
PostgreSQLFr/CS

PostgreSQL : PITR et LogShipping



Jean-Christophe Arnu
PostgreSQLFr/CS

Qu'allons nous manipuler aujourd'hui?

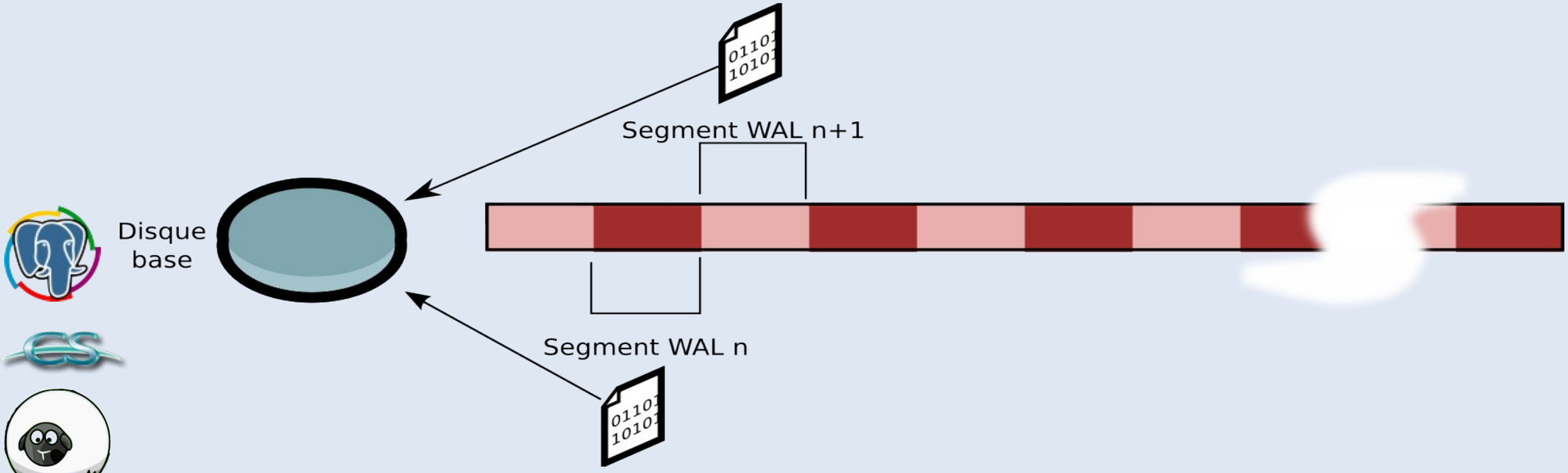
- Rappel des concepts de bases de données
- Qu'est-ce que le PITR?
- Qu'est-ce que le Log Shipping?
- Quand utiliser PITR et LogShipping ?
- Sous quelles conditions?
- Manipulons un peu?

Comment fonctionne une base de données ?

- Des transactions
- Des accès disques (parfois beaucoup)
- Des écritures (parfois souvent)
- Des performances

Améliorer les performances : WAL

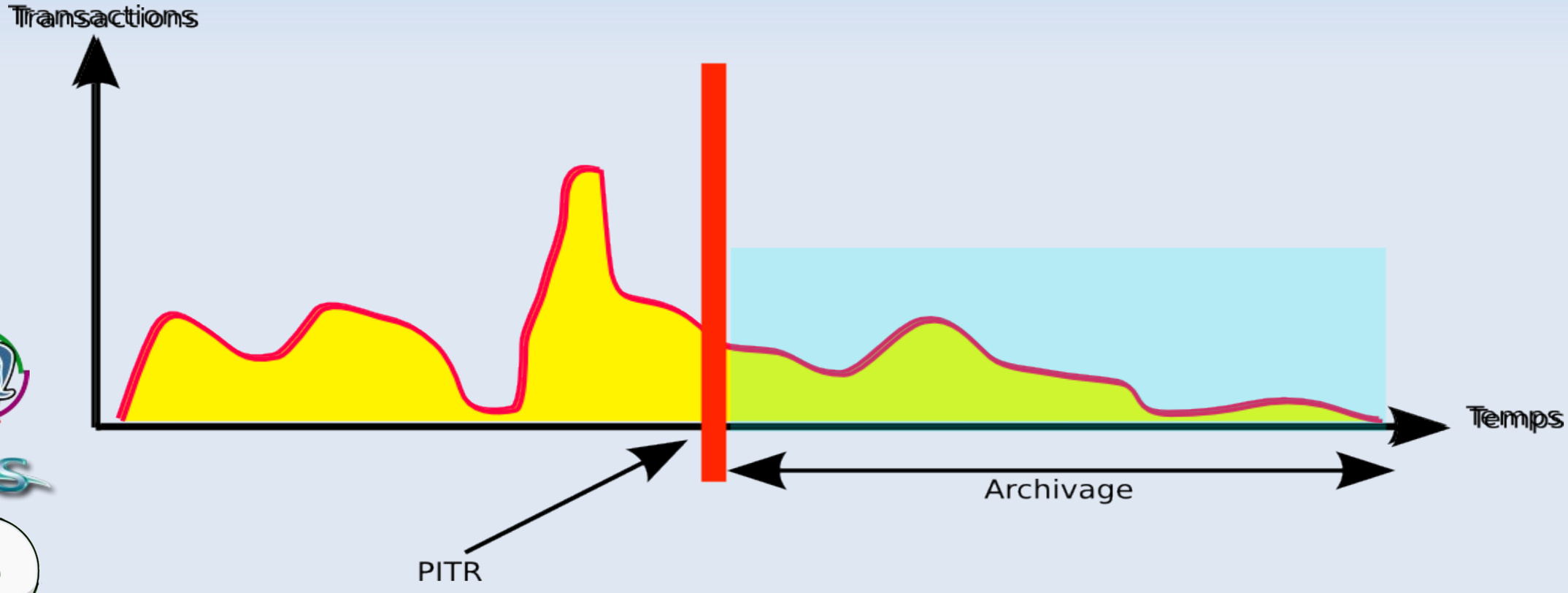
- Écrire dans plusieurs fichiers prend du temps
- Mise à jour d'un journal
- Écriture en différé dans les fichiers des tables



WAL et segments

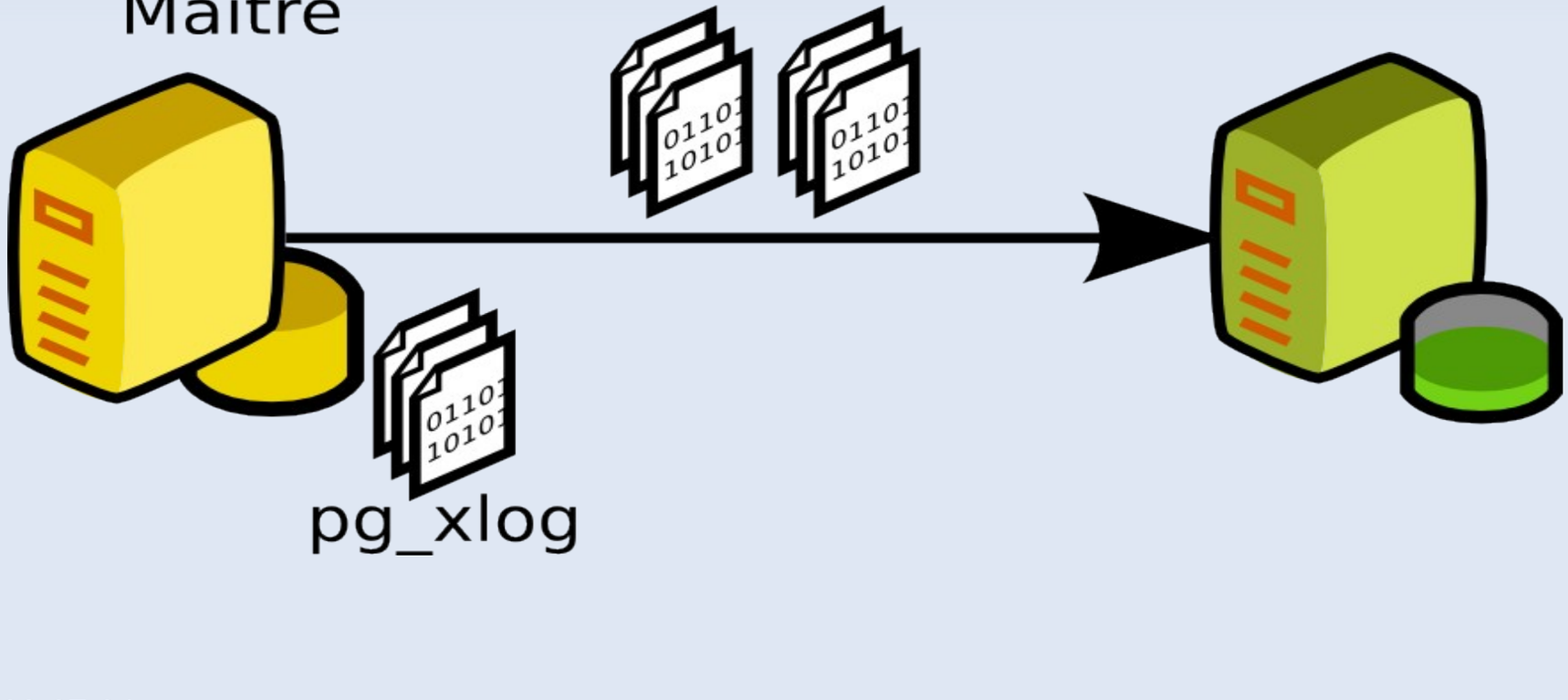
- Un segment WAL = 16Mo
- 3 segments par défaut (`checkpoint_segments`)
- Temps entre deux fichiers WAL : 5 minutes ou `checkpoint_timeout` ou l'appel à `CHECKPOINT`
- Les WAL sont recyclés après écriture de leur contenu dans les fichiers de données de la base

Le PITR



Le LogShipping

- Grande distribution (presque) gratuite de fichiers de journaux transactionnels



Configurer l'archivage du WAL

- Tout se trouve dans le fichier postgresql.conf
  - Debian /etc/postgresql/<version>/<cluster>/
  - Standard /var/lib/pgsql/data
- Quelques options :

```
archive_mode = on          # On active l'archivage
archive_command = '...' # Commande d'archivage
archive_timeout = 60     # Secondes
```

Le script d'archivage

- `archive_command` doit être personnalisé
- %p fichier journal à copier à partir du pg_xlog
- %f nom du fichier (pour la destination)
- A la base :
  - `cp -i %p  /vers/archive/%f`
  - `rsync -a %p login@serveur:/vers/archive/%f`
- Doit retourner 0 en cas de succès et autre chose en cas d'erreur



Gestion de l'archivage

- Ecriture de l'archive en fonction de la sollicitation en `insert`, `update`, `delete`
- Si pas d'activité pour terminer le log, après `archive_timeout` secondes, passage à un autre xlog
- Si indisponibilité du répertoire cible (disque plein, disque injoignable, répertoire absent, ...)
  - Test toutes les secondes (x3)
  - Puis si toujours des erreurs test toutes les minutes




Surveillance

- => Attention à l'augmentation des logs (`pg_xlog` et `pg_xlog/archive_status`)
- Surveiller ces zones (nagios, ...)

Utiliser le PITR

- Sans PITR, il faut arrêter la base pour archiver les fichiers de la base!
- Avec le PITR, on définit un point de repère de la base, on fait un backup, sans arrêter la base!
  - Fonction de préparation au backup à chaud : `pg_start_backup('nom_archive')`
  - Fonction de fin de backup à chaud : `pg_stop_backup()`

pg_start_backup(...)

- Peut prendre un peu de temps pour se terminer
- Effectue un CHECKPOINT pour vider le cache sur disque.
- Crée `$PGDATA/<version>/<cluster>/backup_label`

Sauvegarde des fichiers

- Permet de faire une sauvegarde (tar/rsync) de
  - Sous Debian `$PGDATA/<version>/<cluster>`
  - Sous autre systèmes `$PGDATA`
- Sauvegarde à chaud, les clients peuvent toujours modifier/insérer des données et pas d'influence sur la validité de la sauvegarde

pg_stop_backup()

- Est immédiat.
- Crée `$PGDATA/<version>/<cluster>/backup_label.old`
- On peut, par la suite, se baser sur les fichiers de transactions archivés pour faire de la sauvegarde en continu

Exerçons-nous au PITR

- Déroulement de l'exercice
  - Installation de PostgreSQL
  - Création d'une petite base de test (avec quelques données)
  - Configuration de l'archivage
  - Vérification de l'archivage
  - Création d'une archive
  - Ajout de données
  - Interlude :création d'un autre cluster
  - Autres données




Installation de PostgreSQL / Création de la base initiale

- Installation
  - Installation des paquetages

```
apt-get install postgresql-8.3 postgresql-client-8.3
postgresql-contrib-8.3 postgresql-doc-8.3 python-pgsql wfrench
```

  - Création d'un user base de données :

```
su – postgresql -c createuser pitruser -s -r -d -P -E
```

  - Edition de `/etc/postgresql/8.3/main/pg_hba.conf`

  - Redémarrage

Création de la base

- Création de la base

```
createdb -E UTF-8 pitr-test
```

- Installation du schéma initial

```
psql -u pitruser pitr-test < pitr-schema.sql
```

- Installation de quelques données

```
psql -u pitruser pitr-test < pitr-data.sql
```

- Tests de requêtage

```
psql -u pitruser pitrt-test -c 'SELECT COUNT(*) FROM
moutons;'
```




Configuration de l'archivage

- Edition du fichier /etc/postgresql/8.3/main/postgresql.conf

```
# - Archiving -

archive_mode = on               # allows archiving to be done
                                # (change requires restart)
#archive_command = ''           # command to use to archive a logfile segment
archive_command = 'cp -i %p /var/postgresql/8.3/archive/main/%f < /dev/null'
#archive_timeout = 0            # force a logfile segment switch after this
archive_timeout = 60            # force a logfile segment switch after this
                                # time; 0 is off
```

- Création du répertoire **avec les bons droits!!!**
  - `mkdirhier /var/lib/postgresql/8.3/archive/main/`
  - `chown -R postgres:postgres /var/lib/postgresql/8.3/archive/main`

Vérification de l'archivage

- Relancer le serveur
- Effectuer quelques insertions
  - python logshippingfill.py 5 10
  - Crée 5 bergers ayant chacun 10 moutons

```
SELECT
        b.nom,b.prenom,m.surnom
FROM
        berger b, mouton m, troupeau t
WHERE
        t.berger_id=b.id AND t.mouton_id = m.id;
```

  - Vérifier les archives dans `/var/lib/postgresql/8.3/archive/main`




Création d'une archive

- Démarrer une archive

```
SELECT pg_start_backup('gestion_troupeau_2008-07-04');
SELECT count(*) FROM mouton;
```

- Lancer un archivage avec tar

```
cd /var/postgresql/8.3/main
tar cvjf /var/tmp/archive-main-`date -I`.tar.bz2 .
```

- Arrêter l'archive

```
SELECT pg_stop_backup();
```

Ajout de données

- Relancer le script d'insertion
  - `python logshippingfill.py 5 10`
  - Compter le nombre de moutons
- Noter l'heure : `date -I`
- Relancer à nouveau le script
  - `python logshippingfill.py 10 10`
  - Compter le nombre de moutons



Interlude : création d'un autre cluster

- Il est possible de créer un second cluster:
  - Se comporte comme un serveur à part entière
  - Complètement dissocié du cluster « main »
  - Nouveau cluster « replicat » en tant que root

```
sudo pg_createcluster —start -e UTF-8 8.3 replicat
su - postgres -c 'psql -p 5434 template1 \
   -c "select datname from pg_database"'
  datname
-----------
 template1
 template0
 postgres
(3 lignes)
```



Ajout d'autres données

- L'ajout d'autres données va générer d'autres fichiers dans $PGDATA/main/pg_xlog
- Le répertoire `/var/lib/postgresql/8.3/archive/main` reçoit les journaux de transaction
- Si on ajoute de nouvelles données de nouveaux journaux seront créés
  - Quand un fichier xlog sera plein
  - Quand archive_timeout sera échu
  - Quand pg_switch_xlog() sera appelé



Exercice 2 : restauration d'un PITR

- Arrêt du cluster replicat

```
pg_ctlcluster 8.3 replicat stop
```

- Décompression du fichier tar.bz2 généré précédemment

```
cd /var/postgresql/8.3/replicat
tar xvjf /var/tmp/archive-main-`date -I`.tar.bz2
```

- Redémarrage du cluster et test

```
pg_ctlcluster 8.3 replicat start
psql -p 5434 pitr-test -u pitruser -c 'SELECT count(*) from mouton;'
```

Et la restauration des archives?

- Nous sauvegardons les archives en continu!
- Il est possible de les rejouer sur le cluster : il faut configurer le cluster pour qu'il restaure : `$PGDATA/recovery.conf`
- Exemple : `/usr/share/postgresql/8.3/recovery.conf.sample`

Que se passe-t-il lors de la restauration ?

- Lors la restauration des archives, le serveur est en « warm standby » (inaccessible)
- Lorsque la restauration est terminée, le serveur redevient disponible mais l'apparition de nouveaux fichiers xlog est ignorée
- Une fois la restauration terminée, `$PGDATA/recovery.conf` est renommé `$PGDATA/recovery.done`




Le fichier recovery.conf

- Directives nécessaires
  - `restore_command = 'cp /var/postgresql/8.3/archive/main/%f %p'`
  - Retourne 0 lors d'un succès et <> 0 lors d'un échec
  - `%f` fichier à venir, `%p` répertoire destination, `%r` fichier de redémarrage
- Directives utiles
  - `recovery_target_time` ou `recovery_target_xid`
  - `recovery_target_inclusive`




Exercice 3 : restauration des archives

- Écrire le fichier recovery.conf pour une restauration complète de l'archive
- Surveiller le fichier journal de postgresql (tail -f) dans `/var/log/postgresql/postgresql-8.3-replicat.log`
- Tester la restauration (psql)

Exercice 4 : restauration des archives limitée à un temps

- Utilisation de `recovery_target_time`
- Utiliser l'estampille récupérée toute à l'heure pour ne régénérer que jusqu'à cette date

Le LogShipping

- On conserve le principe de PITR
- On combine avec un transfert des journaux de transactions d'un serveur à l'autre
- On rejoue ces journaux sur l'esclave
- L'esclave est indisponible (*warm-standby*) jusqu'à une erreur d'exécution de `restore_command` (<>0)




pg_standby - 1

- pg_standby permet de disposer d'un outil restore_command intelligent (sur l'esclave)
  - Debug ou journal (-d)
  - Copie ou lien (-c ou -l)
  - Chemin vers le répertoire des fichiers archives
  - Temps d'attente entre deux tentatives (-s <temps en s>)
  - Nombre de tentatives (-r <nombre>)




pg_standby - 2

- Limite du nombre de fichiers dans le répertoire archive (fichiers restants après traitement) (-k <nombre>)
- Temps (désactivable) au bout duquel on rend la main en erreur (-w <temps en s>)
- Fichier dont l'existence arrête l'exécution (-t <file>)

```
pg_standby -d -l -s 2 \
        -t /var/lib/postgresql/8.3/archive/pgsql-8/3-main.trig \
        /var/lib/postgresql/8.3/archive/main %f %d %r
```

Solution intégrée walmgr

- Walmgr est un outil Skype
- Système gérant l'ensemble des fonctions de PITR, d'archivage et de log shipping
- Effectue l'ensemble des taches décrites précédemment au travers d'un seul script python
- S'installe sur maître **et** esclave
- Nécessite l'écriture de fichiers de configuration
- Basé sur rsync, ssh, ...



Mise en place de walmgr - 1

- Fichier de configuration master.ini (/etc/walmagr)

```
master.ini
[wal-master]
logfile                 = master.log
pidfile                 = master.pid
master_db               = dbname=template1
master_data             = /var/lib/postgresql/8.3/main
master_config           = /etc/postgresql/8.3/main/postgresql.conf
slave                   = slave:/var/lib/postgresql/walshipping
slave_config            = /etc/walmgr/slave.ini
completed_wals          = %(slave)s/logs.complete
partial_wals            = %(slave)s/logs.partial
full_backup             = %(slave)s/data.master
loop_delay              = 10.0
use_xlog_functions      = 1
compression             = 1
```




Mise en place de walmgr - 2

- Fichier de configuration de l'esclave (/etc/walmagr)

```
slave.ini
[wal-slave]
logfile                 = slave.log
slave_data              = /var/lib/postgresql/8.3/main
slave_stop_cmd          = /etc/init.d/postgresql-8.3 stop
slave_start_cmd         = /etc/init.d/postgresql-8.3 start
slave                   = /var/lib/postgresql/walshipping
completed_wals          = %(slave)s/logs.complete
partial_wals            = %(slave)s/logs.partial
full_backup             = %(slave)s/data.master
keep_backups            = 5
```




Mise en place de walmgr - 3

- Pour l'utilisateur postgres
  - Permettre les connexions ssh par clé sur l'esclave

```
maitre$ test -f ~/.ssh/id_dsa.pub || ssh-keygen -t dsa
maitre$ cat ~/.ssh/id_dsa.pub | ssh esclave cat \>\> .ssh/authorized_keys
```

  - Configuration et premier archivage sur le maître

```
walmgr.py /chemin/vers/master.ini setup
walmgr.py /chemin/vers/master.ini backup
```

  - Récupération sur l'esclave

```
walmgr.py /chemin/vers/slave.ini restore
```

- Attention pas de restore si pas de backup !!!

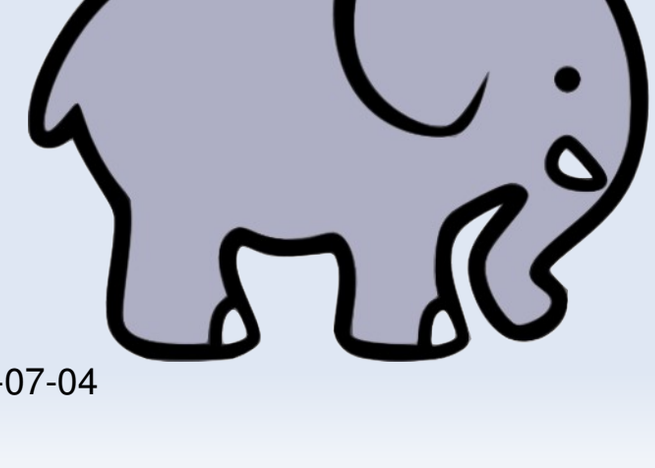


Log shipping avec walmgr

- Envoyer les xlogs
  - Utiliser sync pour un envoi one shot
  - Utiliser syncdaemon pour un envoi au fil de l'eau

PITR et LogShipping

- Et « les moutons seront bien gardés » ;-)
- Le PITR et le log shipping constituent une solution de réplication robuste et fiable
- Des mécanismes simples à mettre en œuvre avec différents outils



Merci

- PostgreSQLFr http://www.postgresqlfr.org
- PostgreSQL http://www.postgresql.org

- Merci
  - Pour être venus
  - Pour m'avoir écouté