

# Réplication de base avec PostgreSQL : le log shipping



par

Date de publication : 14 octobre 2008

Dernière mise à jour :

La version PostgreSQL 8.2 introduit une nouvelle fonctionnalité de haute disponibilité : le **log shipping**. Longtemps réservé aux principaux SGBD du marché, le principe de base miroir est enfin disponible sous PostgreSQL.

I - Introduction.....	3
II - Mise en place du log shipping.....	3
II-A - Installation de Postgresql sur les deux serveurs.....	3
II-B - Installation de pg_standby sur le serveur de secours.....	3
II-C - Initialisation de Postgresql sur le serveur primaire.....	4
II-D - Automatisation du transfert de fichiers depuis le serveur primaire vers le serveur de secours.....	4
II-E - Configuration du serveur primaire pour l'envoi des WALs.....	4
II-F - Initialisation du serveur de secours.....	5
II-G - Synchronisation du serveur de secours.....	6
II-H - Test de bascule sur le serveur de secours.....	7
III - Administration et maintenance de la solution.....	8
III-A - Arrêt et redémarrage de Postgresql sur le serveur primaire.....	8
III-B - Arrêt et redémarrage de Postgresql sur le serveur de secours.....	9
III-C - Echec de l'envoi du WAL sur le serveur de secours.....	9
IV - Conclusion.....	9
V - Annexes.....	10

## I - Introduction

La version Postgresql 8.2 introduit une nouvelle fonctionnalité de haute disponibilité, le **log shipping**. Longtemps réservé aux principaux SGBD du marché, le principe de base miroir est enfin disponible sous Postgresql.

Le but est d'avoir un serveur de secours en attente permanente, qui va rejouer en quasi temps réel les journaux de transaction (appelés WALs pour Write Ahead Log) du serveur primaire, et qui est prêt à prendre le relai à tout moment, en cas d'incident sur le serveur primaire.

Deux serveurs sont donc nécessaires : un serveur primaire (maître) qui archive les WALs et les envoie sur le serveur de secours (esclave), qui lui est en mode recovery permanent et les rejoue dès leur réception. Un lien réseau entre les deux serveurs doit donc exister, pour que l'envoi des WALs puisse se faire.

Cet article détaille les actions nécessaires pour mettre en place le log shipping avec Postgresql 8.2.5 sous Unix, il utilise la contribution **pg\_standby**.

La réplication concerne le cluster Postgresql entier, c'est-à-dire toutes les bases de données Postgresql existantes sur le serveur primaire.

Conventions d'écriture pour cet article :

- sur les deux serveurs, le compte Unix utilisé pour postgresql est 'postgres', avec comme home directory /home/postgres
- la version de Postgresql utilisée est la 8.2.5. Cet article est néanmoins valable pour toutes les versions 8.2.x et supérieures
- le répertoire utilisé pour déposer les sources de Postgresql est /apps/src
- le répertoire utilisé pour installer Postgresql est /apps/postgresql\_8.2.5
- le répertoire utilisé pour initialiser (initdb) Postgresql est /db/data

## II - Mise en place du log shipping

### II-A - Installation de Postgresql sur les deux serveurs

Pour chacun des deux serveurs, télécharger postgresql-8.2.5.tar.gz et le déposer sur dans /apps/src. Compiler ensuite les sources de Postgresql de manière à effectuer l'installation dans le répertoire /apps/postgresql\_8.2.5 :

```
[root@srvX] cd /apps/src
[root@srvX] tar zxvf postgresql-8.2.5.tar.gz
[root@srvX] cd /apps/src/postgresql-8.2.5
[root@srvX] ./configure prefix=/apps/postgresql_8.2.5

[root@srvX] make
# à la fin , le message "All of PostgreSQL successfully made. Ready to install." doit apparaître

[root@srvX] make install
# à la fin , le message "PostgreSQL installation complete." doit apparaître
```

### II-B - Installation de pg\_standby sur le serveur de secours

La contrib pg\_standby sert à surveiller la réception des WALs afin de les rejouer sur le serveur de secours. Elle est constituée de deux fichiers : Makefile et pg\_standby.c, qui sont récupérables sur la page officielle de la contrib [http://anoncvs.postgresql.org/cvsweb.cgi/pgsql/contrib/pg\\_standby](http://anoncvs.postgresql.org/cvsweb.cgi/pgsql/contrib/pg_standby). On déposera ces deux fichiers sur le serveur de secours dans /apps/src :

```
[root@srv2] cd /apps/src
[root@srv2] mkdir -p /apps/src/postgresql-8.2.5/contrib/pg_standby
[root@srv2] mv Makefile /apps/src/postgresql-8.2.5/contrib/pg_standby
```

```
[root@srv2] mv pg_standby.c /apps/src/postgresql-8.2.5/contrib/pg_standby
[root@srv2] cd /apps/src/postgresql-8.2.5/contrib/pg_standby
[root@srv2] make
[root@srv2] make install

# création du répertoire où seront réceptionnés les WALs envoyés par srv1
[root@srv2] su □ postgres
[postgres@srv2] mkdir /db/srv1_wals
```

## II-C - Initialisation de Postgresql sur le serveur primaire

```
[root@srv1] chown -R postgres /db/data
[root@srv1] su □ postgres

[postgres@srv1] /apps/postgresql_8.2.5/bin/initdb -D /db/data
# le message "Success. You can now start the database server using ..." doit apparaître dans les dernières lignes

# répertoire pour stocker les logs applicatifs de postgresql
[postgres@srv1] mkdir □p /db/data/pg_log
```

## II-D - Automatisation du transfert de fichiers depuis le serveur primaire vers le serveur de secours

Il faut maintenant pouvoir automatiser le transfert des WALs du serveur primaire vers le serveur de secours, sans devoir entrer à chaque fois manuellement le mot de passe du compte 'postgres' sur le serveur de secours. On va donc stocker sur ce dernier la clé publique RSA du serveur primaire pour s'affranchir de l'authentification systématique :

```
[postgres@srv1] mkdir □p /home/postgres/.ssh
[postgres@srv1] chmod 700 /home/postgres/.ssh
[postgres@srv1] ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/postgres/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/postgres/.ssh/id_rsa.
Your public key has been saved in /home/postgres/.ssh/id_rsa.pub.
The key fingerprint is:
4e:4b:97:71:5f:d9:b9:d0:1a:0b:fa:7f:af:34:a7:ff postgres@srv1
[postgres@srv1] scp /home/postgres/.ssh/id_rsa.pub postgres@srv2:/home/postgres
Are you sure you want to continue connecting (yes/no)? Yes
postgres@srv2's password:
# saisir ici le mot de passe du compte unix 'postgres' de srv2
```

```
[postgres@srv2] cat /home/postgres/id_rsa.pub >> /home/postgres/.ssh/authorized_keys
[postgres@srv2] rm /home/postgres/id_rsa.pub
[postgres@srv2] chmod 600 /home/postgres/.ssh/authorized_keys
```

Vérifier que l'on peut désormais se connecter depuis srv1 sur srv2 sans demande de mot de passe :

```
[postgres@srv1] ssh postgres@srv2
# on doit maintenant être sur l'invite de commande [postgres@srv2] sans que le mot de passe du compte unix 'postg
```

## II-E - Configuration du serveur primaire pour l'envoi des WALs

Éditer sur le serveur primaire le fichier /db/data/postgresql.conf de manière à avoir les paramètres suivants :

```
archive_command = 'scp %p postgres@srv2:/db/srv1_wals/%f'
archive_timeout
= 300 # on force l'archivage au bout de 300 secondes au maximum quand au moins une transaction a été committée
log_destination = 'stderr'
redirect_stderr = on
log_directory = 'pg_log'
```

```
log_filename = 'postgresql.log'
```

Démarrer Postgresql :

```
[postgres@srv1] /apps/postgresql_8.2.5/bin/pg_ctl start -D /db/data
```

Au bout de quelques dizaines de minutes, vérifier que les WALs sont bien archivés au plus tard toutes les 300 secondes sur le serveur primaire, et envoyés instantanément sur le serveur de secours :

```
[postgres@srv1] ls /db/data/pg_xlog
-rw----- 1 postgres postgres 16777216 Oct 9 16:48 000000010000000000000001
-rw----- 1 postgres postgres 16777216 Oct 9 16:53 000000010000000000000002
-rw----- 1 postgres postgres 16777216 Oct 9 16:43 000000010000000000000003
[postgres@srv2] ls -lrt /db/srv1_wals
-rw----- 1 postgres postgres 16777216 Oct 9 16:42 000000010000000000000000
-rw----- 1 postgres postgres 16777216 Oct 9 16:47 000000010000000000000001
-rw----- 1 postgres postgres 16777216 Oct 9 16:52 000000010000000000000002
```

Dans cet exemple, le WAL 000000010000000000000003 est le WAL courant sur srv1, il n'a donc pas encore été archivé ni envoyé sur srv2.

## II-F - Initialisation du serveur de secours

Il faut maintenant initialiser le serveur de secours à partir d'une sauvegarde à chaud du serveur primaire.

Mettre le Postgresql en mode backup sur le serveur primaire, puis recopier l'arborescence de chaque tablespace sur le serveur de secours, au même emplacement.

Sur srv1 :

```
[postgres@srv1] /apps/postgresql_8.2.5/bin/psql -c "select pg_start_backup('pour_init_srv2')"
```

```
pg_start_backup
-----
0/4000020
(1 row)
```

```
[postgres@srv1] tar zcvf /tmp/bkp_pour_init_srv2.tar.gz /db/data
```

```
[postgres@srv1] /apps/postgresql_8.2.5/bin/psql -c "select pg_stop_backup()"
```

```
pg_stop_backup
-----
0/4000084
(1 row)
```

```
[postgres@srv1] scp /tmp/bkp_pour_init_srv2.tar.gz postgres@srv2:/tmp
```

Sur srv2 :

```
[postgres@srv2] cd /
[postgres@srv2] tar zxvf /tmp/bkp_pour_init_srv2.tar.gz
```

Dans l'exemple ci-dessus, on considère que toutes les données des bases Postgresql sont contenues dans le chemin /db/data, et qu'il n'y a aucun autre tablespace situé sur un autre disque (sinon il suffirait de faire un 'tar' de l'arborescence de chaque tablespace puis de la recopier au même endroit sur le serveur de secours)

Éditer sur le serveur de secours le fichier /db/data/postgresql.conf de manière à avoir les paramètres suivants (l'archivage pouvant être désactivé sur la base de secours) :

```
archive_command = '' # double quote
archive_timeout = 0
log_destination = 'stderr'
log_directory = 'pg_log'
log_filename = 'postgresql.log'
redirect_stderr = on
```

Créer sur le serveur de secours un fichier `/db/data/recovery.conf` (pour `pg_standby`, ce fichier doit être à la racine du répertoire `/db/data`) contenant la ligne suivante (toute la commande est sur une seule ligne) :

```
restore_command=' /apps/postgresql_8.2.5/bin/pg_standby -d -k 255 -t /home/postgres/stoprestore.file /db/srv1_wals/ %f %p 2>> /db/pg_standby.log'
```

Ainsi, le fichier `/db/pg_standby.log` contiendra en permanence les informations sur les WALs rejoués, et celui en attente d'être rejoué. Pour stopper la réplication, il suffira simplement de créer sur le serveur de secours le fichier `/home/postgres/stoprestore.file`

## II-G - Synchronisation du serveur de secours

Démarrer Postgresql sur le serveur de secours, il se mettra automatiquement en mode recovery, étant donné la présence du fichier `/db/data/recovery.conf` :

```
[postgres@srv2] /apps/postgresql_8.2.5/bin/pg_ctl start -D /db/data
pg_ctl: another server might be running; trying to start server anyway
server starting
```

Noter la présence de processus liés à `pg_standby`, qui scrutent en permanence l'arrivée des WALs pour les rejouer :

```
[postgres@srv2 /]$ ps -fu postgres | grep pg_standby | grep □v grep
```

Vérifier dans le fichier `/db/pg_standby.log` que les WALs envoyés par le serveur primaire dans `/db/srv1_wals` sont bien rejoués :

```
[postgres@srv2] tail -50 /db/pg_standby.log
WAL file not present yet. Checking for trigger file...
WAL file not present yet. Checking for trigger file...
running restore : OK
Trigger file : /home/postgres/stoprestore.file
Waiting for WAL file : 00000001000000000000000F
WAL file path : /db/srv1_wals//000000010000000000000007
Restoring to... : pg_xlog/RECOVERYXLOG
Sleep interval : 5 seconds
Max wait interval : 0 forever
Command for restore : cp "/db/srv1_wals//000000010000000000000007" "pg_xlog/RECOVERYXLOG"
Keep archive history : No cleanup required
WAL file not present yet. Checking for trigger file...
WAL file not present yet. Checking for trigger file...
WAL file not present yet. Checking for trigger file...
WAL file not present yet. Checking for trigger file...
running restore : OK
Trigger file : /home/postgres/stoprestore.file
Waiting for WAL file : 00000001000000000000000F
WAL file path : /db/srv1_wals//000000010000000000000008
Restoring to... : pg_xlog/RECOVERYXLOG
Sleep interval : 5 seconds
Max wait interval : 0 forever
Command for restore : cp "/db/srv1_wals//000000010000000000000008" "pg_xlog/RECOVERYXLOG"
Keep archive history : No cleanup required
WAL file not present yet. Checking for trigger file...
WAL file not present yet. Checking for trigger file...
```

Le serveur de secours est donc maintenant opérationnel et rejoue en permanence les WALs envoyés par le serveur primaire.

## II-H - Test de bascule sur le serveur de secours

Simulons maintenant une panne sur le serveur primaire, et vérifions la perte de données lors de la bascule sur le serveur de secours.

On va créer pour cela sur le serveur primaire une table t1 contenant une ligne, forcer ensuite un archivage du WAL courant (et donc son envoi sur le serveur de secours pour réplication), créer ensuite une deuxième table t2 contenant une ligne, puis désactiver immédiatement après l'envoi des WALs, afin que le WAL courant ayant enregistré les transactions relatives à la table t2 n'ait pu être envoyé à temps sur le serveur de secours. Enfin, nous allons activer le serveur de secours afin de vérifier la réplication.

```
[postgres@srv1] /apps/postgresql_8.2.5/bin/psql
postgres=# create table t1 (c varchar(5));
CREATE TABLE
postgres=# insert into t1 values ('test1');
INSERT 0 1
postgres=# select pg_switch_xlog();
pg_switch_xlog
-----
 0/C0117F8
(1 row)
postgres=# create table t2 (c varchar(5));
CREATE TABLE
postgres=# insert into t2 values ('test2');
INSERT 0 1
```

À noter que la fonction `pg_switch_log()` n'archivera pas le WAL courant si celui-ci ne contient aucune transaction committée depuis l'archivage du WAL précédent.

Sur le serveur primaire, éditer rapidement (avant l'archivage du WAL suivant) le fichier `/db/data/postgresql.conf` pour positionner le paramètre `archive_command` de la manière suivante :

```
archive_command='' # double quote pour le désactiver
```

Recharger ensuite immédiatement la configuration Postgresql du serveur primaire pour prendre en compte à chaud la nouvelle valeur du paramètre :

```
[postgres@srv1] /apps/postgresql_8.2.5/bin/pg_ctl reload -D /db/data
server signaled
```

À partir de maintenant, les WALs ne sont plus envoyés au serveur de secours. C'est ce qui se produit lors d'une panne sur le serveur primaire.

Activons maintenant le serveur de secours pour évaluer la perte de données :

```
[postgres@srv2] touch /home/postgres/stoprestore.file
# on constate au bout de quelques secondes que le fichier /db/data/recovery.conf a été renommé en /db/
data/recovery.done
```

```
[postgres@srv2] tail -15 /db/pg_standby.log
WAL file not present yet. Checking for trigger file...
WAL file not present yet. Checking for trigger file...
WAL file not present yet. Checking for trigger file...
WAL file not present yet. Checking for trigger file...
WAL file not present yet. Checking for trigger file...trigger file found
Trigger file : /home/postgres/stoprestore.file
Waiting for WAL file : 000000010000000000000000
WAL file path : /db/srv1_wals//000000010000000000000000
Restoring to... : pg_xlog/RECOVERYXLOG
Sleep interval : 5 seconds
Max wait interval : 0 forever
```

```
Command for restore : cp "/db/srv1_wals//000000010000000000000000C" "pg_xlog/RECOVERYXLOG"  
Keep archive history : No cleanup required  
running restore : OK
```

```
[postgres@srv2] tail /db/data/pg_log/postgresql.log  
...  
LOG: archive recovery complete  
LOG: database system is ready
```

Du fait de la présence du fichier /home/postgres/stoprestore.file, la réplication a été interrompue, et le serveur de secours est maintenant activé : il est désormais possible de se connecter à Postgresql.

```
[postgres@srv2] /apps/postgresql_8.2.5/bin/psql  
postgres=# select * from t1;  
c  
-----  
test1  
(1 row)  
  
postgres=# select * from t2;  
ERROR: relation "t2" does not exist
```

On a donc bien la confirmation que la table t1 a été répliquée sur le serveur de secours, car le WAL a bien été envoyé, lorsqu'on a forcé l'archivage via la commande `pg_switch_xlog()`. Par contre, les transactions concernant la création de la table t2 et l'insertion d'une ligne dans cette table étant dans le WAL courant, non envoyé au moment de la panne, elles n'ont pas pu être rejouées sur le serveur de secours.

Nous pouvons donc en conclure que le log shipping effectue une réplication en quasi temps réel, c'est-à-dire qu'il existe toujours un décalage lié au WAL courant de la base primaire, qui n'est pas encore archivé. Lorsqu'une panne se produit sur le serveur primaire, seules les transactions stockées dans le WAL courant du serveur primaire peuvent ne pas être répliquées sur le serveur de secours si ce WAL n'a pas pu être envoyé.

C'est pourquoi il convient de forcer sur le serveur primaire un archivage fréquent du WAL courant, afin de limiter la perte de données en cas de bascule. Dans l'exemple, la perte maximale de données était de 300 secondes (correspondant au paramètre `archive_timeout` du fichier /db/data/postgresql.conf)

### III - Administration et maintenance de la solution

Les deux serveurs sont complètement indépendants et n'ont pas besoin l'un de l'autre pour fonctionner, c'est pourquoi il est tout-à-fait possible d'arrêter/redémarrer Postgresql sur chacun des deux serveurs pour des opérations de maintenance (reboot, changement de paramètres non dynamiques du fichier `postgresql.conf`, □), ou d'autoriser une coupure temporaire du réseau qui empêcherait l'envoi du WAL.

La réplication reprendra automatiquement dès que la situation normale sera rétablie.

#### III-A - Arrêt et redémarrage de Postgresql sur le serveur primaire

```
[postgres@srv1] /apps/postgresql_8.2.5/bin/pg_ctl stop -D /db/data  
[postgres@srv1] /apps/postgresql_8.2.5/bin/pg_ctl start -D /db/data
```

Quand le serveur primaire est arrêté, le serveur de secours ne le sait pas et reste quand-même en attente du WAL suivant

Le serveur primaire recommencera à envoyer les WAL sur le serveur de secours dès que la situation normale sera rétablie.

### III-B - Arrêt et redémarrage de Postgresql sur le serveur de secours

```
[postgres@srv2] /apps/postgresql_8.2.5/bin/pg_ctl stop -D /db/data -m fast
[postgres@srv2] /apps/postgresql_8.2.5/bin/pg_ctl start -D /db/data
```

Si plusieurs WAL non-répliqués sont présents dans le répertoire /db/srv1\_wals quand Postgresql est redémarré sur le serveur de secours, la réplication reprendra automatiquement depuis le dernier WAL rejoué.

### III-C - Echec de l'envoi du WAL sur le serveur de secours

Pour cause d'indisponibilité du serveur de secours ou de coupure temporaire sur le réseau, il se peut que l'envoi du WAL (la commande 'scp') depuis le serveur primaire échoue. Dans ce cas, aucune action n'est nécessaire, le serveur primaire réessaiera d'envoyer le WAL jusqu'à ce qu'il réussisse, lorsque le problème sera résolu. Tous les WALs non envoyés durant cette période le seront automatiquement dès la coupure rétablie et la réplication reprendra, sans incidence sur le serveur de secours (celui-ci aura simplement attendu les WALs durant la coupure).

#### Pendant l'échec d'envoi des WAL :

```
[postgres@srv1] tail /db/data/pg_log/postgresql.log
ssh: srv2: Name or service not known
lost connection
LOG: archive command "scp pg_xlog/000000010000000000000001F
postgres@srv2:/db/srv1_wals/000000010000000000000001F" failed: return code 256
ssh: tes0046: Name or service not known
lost connection
LOG: archive command "scp pg_xlog/000000010000000000000001F
postgres@srv2:/db/srv1_wals/000000010000000000000001F" failed: return code 256
WARNING: transaction log file "000000010000000000000001F" could not be archived: too many failures
```

#### Une fois le problème résolu :

```
[postgres@srv1] tail /db/data/pg_log/postgresql.log
lost connection
LOG: archive command "scp pg_xlog/000000010000000000000001F
postgres@tes0046:/datas/pgdatas/srv1_wals/000000010000000000000001F" failed: return code 256
WARNING: transaction log file "000000010000000000000001F" could not be archived: too many failures
LOG: received SIGHUP, reloading configuration files
LOG: archived transaction log file "000000010000000000000001F"
LOG: archived transaction log file "0000000100000000000000020"
LOG: archived transaction log file "0000000100000000000000021"
LOG: archived transaction log file "0000000100000000000000022"
LOG: archived transaction log file "0000000100000000000000023"
LOG: archived transaction log file "0000000100000000000000024"
```

## IV - Conclusion

Le log shipping est une fonctionnalité très importante de Postgresql depuis la version 8.2. Sa mise en place est assez simple et permet d'assurer une haute disponibilité des données, ce qui, de nos jours, peut s'avérer déterminant dans le choix de départ du SGBD pour des applications sensibles.

La perte de données en cas de bascule peut être réduite à quelques minutes seulement, ce qui, en cas de panne, est souvent préférable plutôt que de perdre plusieurs heures à remonter un autre serveur, réinstaller Postgresql, restaurer la dernière sauvegarde des données et rejouer les WALs jusqu'à l'instant précédent l'incident.

### Contrairement à d'autres SGBD proposant cette fonctionnalité, quelques limitations existent néanmoins sous PostgreSQL :

- Il est impossible d'échanger temporairement les rôles des deux serveurs, c'est-à-dire que le serveur primaire devienne serveur de secours et vice-versa (action communément appelée "switchover"). Une fois la bascule sur le serveur de secours effectuée, celui-ci devient donc serveur primaire. S'il l'on souhaite recréer un serveur de secours, il faut reprendre toutes les étapes de cet article depuis le début.

- Tant que le serveur de secours est en mode recovery, il est impossible de se connecter à Postgresql. Le serveur de secours ne peut donc pas être utilisé en lecture seule dans le but de décharger le serveur primaire de certaines requêtes ou activités de reporting des utilisateurs.

## V - Annexes

Documentation officielle sur :  <http://www.postgresql.org/docs/>