Module I4

Outils de sauvegarde physique

DALIBO
L'expertise PostgreSQL

22.09

Dalibo SCOP

https://dalibo.com/formations

Outils de sauvegarde physique

Module I4

TITRE : Outils de sauvegarde physique
SOUS-TITRE : Module I4

REVISION: 22.09
DATE: 02 septembre 2022

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

LICENCE CREATIVE COMMONS BY-NC-SA 2.0 FR

Attribution - Pas d'Utilisation Commerciale - Partage dans les Mêmes Conditions

Vous êtes autorisé à :

- Partager, copier, distribuer et communiquer le matériel par tous moyens et sous tous formats
- Adapter, remixer, transformer et créer à partir du matériel

Dalibo ne peut retirer les autorisations concédées par la licence tant que vous appliquez les termes de cette licence selon les conditions suivantes :

Attribution : Vous devez créditer l'œuvre, intégrer un lien vers la licence et indiquer si des modifications ont été effectuées à l'œuvre. Vous devez indiquer ces informations par tous les moyens raisonnables, sans toutefois suggérer que Dalibo vous soutient ou soutient la façon dont vous avez utilisé ce document.

Pas d'Utilisation Commerciale : Vous n'êtes pas autorisé à faire un usage commercial de ce document, tout ou partie du matériel le composant.

Partage dans les Mêmes Conditions : Dans le cas où vous effectuez un remix, que vous transformez, ou créez à partir du matériel composant le document original, vous devez diffuser le document modifié dans les même conditions, c'est à dire avec la même licence avec laquelle le document original a été diffusé.

Pas de restrictions complémentaires : Vous n'êtes pas autorisé à appliquer des conditions légales ou des mesures techniques qui restreindraient légalement autrui à utiliser le document dans les conditions décrites par la licence.

Note : Ceci est un résumé de la licence. Le texte complet est disponible ici :

https://creativecommons.org/licenses/by-nc-sa/2.0/fr/legalcode

Pour toute demande au sujet des conditions d'utilisation de ce document, envoyez vos questions à contact@dalibo.com[1] !

[1] mailto:contact@dalibo.com

Chers lectrices & lecteurs,

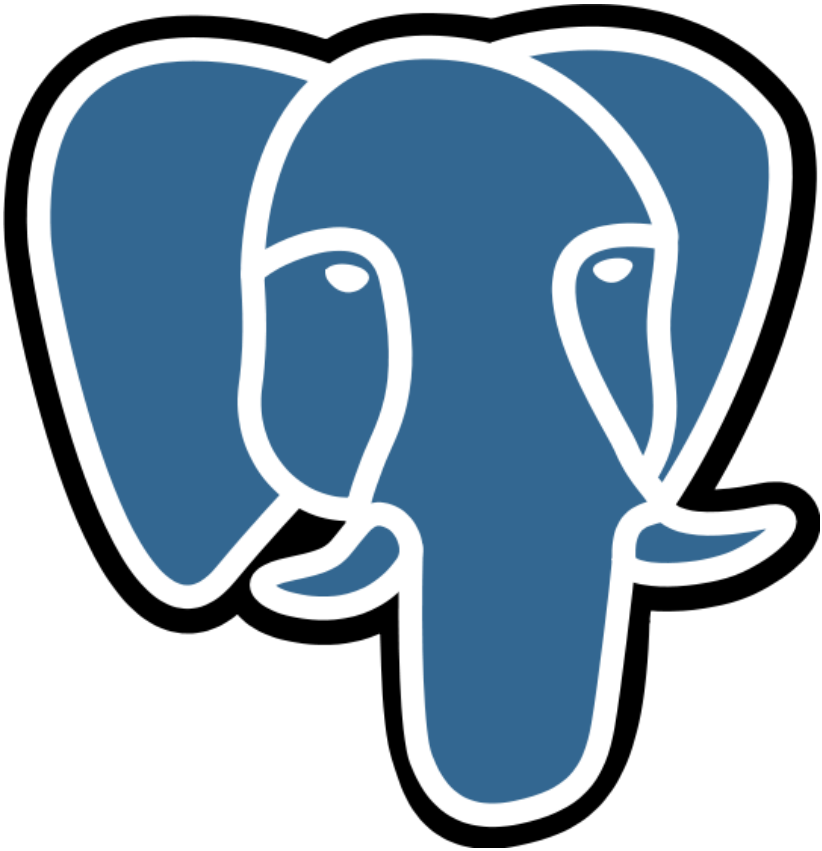
Nos formations PostgreSQL sont issues de nombreuses années d'études, d'expérience de terrain et de passion pour les logiciels libres. Pour Dalibo, l'utilisation de PostgreSQL n'est pas une marque d'opportunisme commercial, mais l'expression d'un engagement de longue date. Le choix de l'Open Source est 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 !

Table des Matières

1 POSTGRESQL : OUTILS DE SAUVEGARDE PHYSIQUE



1.1 INTRODUCTION

- 2 mécanismes de sauvegarde natifs et robustes
- Industrialisation fastidieuse
- Des outils existent !!

Nous avons vu le fonctionnement interne du mécanisme de sauvegarde physique. Celui-ci étant en place nativement dans le moteur PostgreSQL depuis de nombreuses versions, sa robustesse n'est plus à prouver. Cependant, son industrialisation reste fastidieuse.

Des outils tiers existent et vont permettre de faciliter la gestion des sauvegardes, de leur mise en place jusqu'à la restauration. Dans ce module nous allons voir en détail certains de ces outils et étudier les critères qui vont nous permettre de choisir la meilleure solution selon notre contexte.

1.1.1 AU MENU

- Présentation:
  - pg_basebackup
  - pgBackRest
  - Barman
  - pitrery
- Comment choisir ?

Lors de cette présentation, nous allons passer en revue les différents outils principaux de gestion de sauvegardes, leurs forces, le paramétrage, l'installation et l'exploitation.

1.1.2 DÉFINITION DU BESOIN - CONTEXTE

- Sauvegarde locale (ex. NFS) ?
- Copie vers un serveur tiers (push) ?
- Sauvegarde distante initiée depuis un serveur tiers (pull) ?
- Ressources à disposition ?
- Accès SSH ?
- OS ?
- Sauvegardes physiques ? Logiques ?
- Version de PostgreSQL ?
- Politique de rétention ?

Où les sauvegardes doivent-elles être stockées ?

Quelles ressources sont à disposition : serveur de sauvegarde dédié ? quelle puissance pour la compression ?

De quel type d'accès aux serveurs de base de données dispose-t-on ? Quelle est la version du système d'exploitation ?

Il est très important de se poser toutes ces questions, les réponses vont servir à établir le contexte et permettre de choisir l'outil et la méthode la plus appropriée.

Attention, pour des raisons de sécurité et de fiabilité, les répertoires choisis pour la restauration des données de votre instance **ne doivent pas** être à la racine d'un point de montage.
Si un ou plusieurs points de montage sont dédiés à l'utilisation de PostgreSQL, positionnez toujours les données dans un sous-répertoire, voire deux niveaux en dessous du point de montage (eg. `<point de montage>/<version majeure>/<nom instance>`).

1.2 PG_BASEBACKUP - PRÉSENTATION

- Outil intégré à PostgreSQL
- Prévu pour créer une instance secondaire
- Pour sauvegarde ponctuelle
  - PITR avec outils complémentaires

pg_basebackup[2] est une application cliente intégrée à PostgreSQL, au même titre que pg_dump ou pg_dumpall.

pg_basebackup a été conçu pour permettre l'initialisation d'une instance secondaire, et il peut donc être utilisé pour effectuer facilement une sauvegarde physique ponctuelle. Celle-ci inclut les fichiers et journaux nécessaires pour une restauration telle que l'instance était à la fin de la sauvegarde.

pg_basebackup peut aussi être à la base d'outils permettant le PITR (par exemple barman). Ces outils s'occupent en plus de l'archivage des journaux générés pendant et après la sauvegarde initiale, pour une restauration dans un état postérieur à la fin de cette sauvegarde.

1.2.1 PG_BASEBACKUP - FORMATS DE SAUVEGARDE

- `--format plain`
  - arborescence identique à l'instance sauvegardée
- `--format tar`
  - archive
  - compression : `-z`, `-Z (0..9)`

Le format par défaut de la sauvegarde est `plain`, ce qui signifie que les fichiers seront créés tels quels dans le répertoire de destination (ou les répertoires en cas de tablespaces). C'est idéal pour obtenir une copie immédiatement utilisable.

[2] https://www.postgresql.org/docs/current/static/app-pgbasebackup.html

Pour une archive à proprement parler, préférer l'option `--format tar`. pg_basebackup génère alors une archive `base.tar` pour le PGDATA de l'instance, puis une archive `<oid>.tar` par tablespace. Les journaux récupérés seront également dans un fichier `.tar`.

L'option `--gzip` (`-z`) ajoute la compression `gzip`. Le niveau de compression peut également être spécifié avec `--compress=1` à `9` (`-Z`). Cela permet d'arbitrer entre la durée de la sauvegarde et sa taille.

1.2.2 PG_BASEBACKUP - AVANTAGES

- Transfert des WAL pendant la sauvegarde
- Slot de réplication automatique (temporaire voire permanent)
- Limitation du débit
- Relocalisation des tablespaces
- Fichier manifeste (v13+)
- Vérification des checksums (v11+)
- Sauvegarde possible à partir d'un secondaire
- Suivi : `pg_stat_progress_basebackup` (v13+)

pg_basebackup s'est beaucoup amélioré au fil des versions et son comportement a parfois changé. Regardez bien la documentation[3] de votre version.

Même avec un serveur un peu ancien, il possible d'installer un `pg_basebackup` récent, en installant les outils clients de la dernière version de PostgreSQL.

Récupération des journaux :

pg_basebackup sait récupérer les fichiers WAL nécessaires à la restauration de la sauvegarde sans passer par la commande d'archivage. Il connaît deux méthodes :

Avec l'option `--wal-method fetch` (ou `-X`), les WAL générés pendant la sauvegarde seront demandés une fois celle-ci terminée, à condition qu'ils n'aient pas été recyclés entretemps (ce qui peut nécessiter un slot de réplication, ou éventuellement une configuration élevée du paramètre `wal_keep_size`/`wal_keep_segments`).

L'option par défaut depuis PostgreSQL 10 est cependant `-X stream` : les WAL sont récupérés non pas en fin de sauvegarde, mais *en streaming* pendant celle-ci. Cela nécessite néanmoins l'utilisation d'un *wal sender* supplémentaire, le paramètre `max_wal_senders` doit parfois être augmenté en conséquence.

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

Rappelons que si l'archivage des WAL n'est pas actif, la sauvegarde effectuée ne sera utilisée que pour restaurer l'instance telle qu'elle était au moment de la fin de la sauvegarde : il ne sera pas possible de réaliser une restauration PITR.

À l'inverse, `-X none` peut être utile si la récupération des journaux est réalisée par ailleurs (généralement par `archive_command`). Attention, l'archive réalisée avec pg_basebackup ne sera alors pas « complète », et ne pourra pas être restaurée sans ces archives des journaux (il faudra indiquer où aller les chercher avec `restore_command`.)

Slots de réplication :

Par défaut, pg_basebackup va créer un slot de réplication temporaire sur le serveur pour sécuriser la sauvegarde. Il disparaîtra une fois celle-ci terminée.

Pour faciliter la mise en place d'une instance secondaire, et garantir que tous les journaux nécessaires seront encore sur le primaire à son démarrage, il est possible de créer un slot de réplication permanent, et de le fournir à pg_basebackup avec `--slot nom_du_slot`. pg_basebackup peut le créer lui-même avec `--create`. Si l'on préfère le créer préalablement, il suffit d'exécuter la requête suivante :

```
SELECT pg_create_physical_replication_slot ('nom_du_slot');
```

Rappelons qu'un slot inutilisé doit être rapidement supprimé pour ne pas mener à une dangereuse accumulation des journaux.

Sécurisation de la sauvegarde :

Par défaut, `pg_basebackup` crée un fichier manifeste (à partir de PostgreSQL 13). Ce fichier contient la liste des fichiers sauvegardés, leur taille et leur somme de contrôle. Cela permet après coup de vérifier l'intégrité de la sauvegarde à l'aide de l'outil `pg_verifybackup`.

L'algorithme par défaut de la somme de contrôle, CRC32, suffit pour détecter une erreur technique accidentelle ; d'autres algorithmes disponibles permettent de détecter une manipulation volontaire de la sauvegarde.

Vérification des sommes de contrôle :

À partir de la version 11, une sauvegarde avec pg_basebackup entraîne la vérification des sommes de contrôle de l'instance. Cela garantit que la sauvegarde n'héritera pas d'une corruption existante, sinon l'outil tombe en erreur.

L'option `--no-verify-checksums` autorise la sauvegarde d'une instance où une corruption est détectée (sauvegarde aussi problématique, certes, mais qui peut permettre de travailler sur la récupération, ou de sauver l'essentiel).

Autres options :

Le débit de la sauvegarde est configurable avec l'option `--max-rate=` (`-r`) pour limiter l'impact sur l'instance ou le réseau. Cette restriction de débit ne concerne pas les journaux transférés en parallèle (`-X stream`).

Pour gagner un peu de temps, si l'instance n'est pas trop chargée, `--checkpoint=fast` accélère le checkpoint préalable à la sauvegarde.

Avec une sauvegarde `plain`, il est possible de modifier sur la cible les chemins des éventuels tablespaces avec l'option `--tablespace-mapping=<vieuxrep>=<nouveaurep>` (ou `-T`), et de relocaliser le répertoire des fichiers WAL avec l'option `--waldir=<nouveau chemin>`.

Depuis un secondaire :

pg_basebackup permet nativement de réaliser une sauvegarde à partir d'une instance secondaire. Le paramétrage nécessaire figure plus bas.

Suivi :

Pour suivre le déroulement de la sauvegarde depuis un terminal, il existe l'option `--progress` (`-P`).

À partir de PostgreSQL 13, il existe aussi une vue pour ce suivi : `pg_stat_progress_basebackup`[4] .

Options complètes :

Pour mémoire, toutes les options disponibles sont celles-ci (en version 14) :

```
$ pg_basebackup --help
pg_basebackup prend une sauvegarde binaire d'un serveur PostgreSQL en cours
d'exécution.

Usage :
  pg_basebackup [OPTION]...

Options contrôlant la sortie :
  -D, --pgdata=RÉPERTOIRE        reçoit la sauvegarde de base dans ce répertoire
  -F, --format=p|t               format en sortie (plain (par défaut), tar)
  -r, --max-rate=TAUX            taux maximum de transfert du répertoire de
                                 données (en Ko/s, ou utiliser le suffixe « k »
                                 ou « M »)
  -R, --write-recovery-conf      écrit la configuration pour la réplication
  -T, --tablespace-mapping=ANCIENREP=NOUVEAUREP
                                 déplace le répertoire ANCIENREP en NOUVEAUREP
```

[4] https://docs.postgresql.fr/current/progress-reporting.html#BASEBACKUP-PROGRESS-REPORTING

```
      --waldir=RÉP_WAL           emplacement du répertoire des journaux de
                                 transactions
  -X, --wal-method=none|fetch|stream
                                 inclut les journaux de transactions requis avec
                                 la méthode spécifiée
  -z, --gzip                     compresse la sortie tar
  -Z, --compress=0-9             compresse la sortie tar avec le niveau de
                                 compression indiqué

Options générales :
  -c, --checkpoint=fast|spread   exécute un CHECKPOINT rapide ou réparti
      --create-slot              crée un slot de réplication
  -l, --label=LABEL              configure le label de sauvegarde
  -n, --no-clean                 ne nettoie pas en cas d'erreur
  -N, --no-sync                  n'attend pas que les modifications soient
                                 proprement écrites sur disque
  -P, --progress                 affiche la progression de la sauvegarde
  -S, --slot=NOMREP              slot de réplication à utiliser
  -v, --verbose                  affiche des messages verbeux
  -V, --version                  affiche la version puis quitte
      --manifest-checksums=SHA{224,256,384,512}|CRC32C|NONE
                                 utilise cet algorithme pour les sommes de
                                 contrôle du manifeste
      --manifest-force-encode    encode tous les noms de fichier dans le
                                 manifeste en hexadécimal
      --no-estimate-size         ne réalise pas d'estimation sur la taille de la
                                 sauvegarde côté serveur
      --no-manifest              supprime la génération de manifeste de
                                 sauvegarde
      --no-slot                  empêche la création de slots de réplication
                                 temporaires
      --no-verify-checksums      ne vérifie pas les sommes de contrôle
  -?, --help                     affiche cette aide puis quitte

Options de connexion :
  -d, --dbname=CHAÎNE_CONNEX     chaîne de connexion
  -h, --host=HÔTE                hôte du serveur de bases de données ou
                                 répertoire des sockets
  -p, --port=PORT                numéro de port du serveur de bases de données
  -s, --status-interval=INTERVAL durée entre l'envoi de paquets de statut au
                                 serveur (en secondes)
  -U, --username=UTILISATEUR     se connecte avec cet utilisateur
  -w, --no-password              ne demande jamais le mot de passe
  -W, --password                 force la demande du mot de passe (devrait
                                 survenir automatiquement)
```

```
Rapporter les bogues à <pgsql-bugs@lists.postgresql.org>.
Page d'accueil de PostgreSQL : <https://www.postgresql.org/>
```

1.2.3 PG_BASEBACKUP - LIMITATIONS

- Configuration *streaming* nécessaire
- Pas de configuration de l'archivage
- Pas d'association WAL archivés / sauvegarde
- Pas de politique de rétention
  - sauvegarde ponctuelle
- Pas de gestion de la restauration !
  - manuel : `recovery.signal`, `restore_command`…
  - pour un secondaire : `--write-recovery-conf`

Configuration :

pg_basebackup étant conçu pour la mise en place d'une instance en réplication, l'instance principale nécessite d'être configurée en conséquence :

- `max_wal_senders` doit avoir une valeur supérieure à `0` pour permettre à pg_basebackup de se connecter (au moins `2` si on utilise le transfert des WAL par streaming) — c'est le cas par défaut ;
- le fichier `pg_hba.conf` de l'instance principale doit être configuré pour autoriser les connexions de type `replication` depuis la machine où la sauvegarde est déclenchée, par exemple ainsi :

```
host  replication  repli_user  192.168.0.100/32  scram-sha-256
```

Dans l'idéal, l'utilisateur employé est dédié à la réplication. Pour automatiser, stocker le mot de passe nécessaire dans un fichier `.pgpass`.

L'archivage n'est pas géré par pg_basebackup. Il ne récupère par *streaming* que les journaux nécessaires à la cohérence de sa sauvegarde. Il faudra paramétrer `archive_command` à la main pour une sauvegarde PITR.

Si la sauvegarde est effectuée à partir d'une instance secondaire :

- ces paramétrages sont nécessaires (et en place par défaut) :
  - instance secondaire ouverte en lecture (`hot_standby` à `on`) ;
  - `max_wal_senders` supérieur `0` et droits en place pour permettre à pg_basebackup de se connecter ;
  - écriture complète des pages dans les WAL activée (`full_page_writes` à `on`) ;

- pour du PITR :
  - l'archivage des fichiers WAL doit être configuré indépendamment.
  - une attention particulière doit être apportée au fait que tous les fichiers WAL nécessaires à la restauration ont bien été archivés.

Gestion des sauvegardes :

La gestion des sauvegardes (rétention, purge…) n'est pas prévue dans l'outil.

pg_basebackup n'effectue pas non plus de lien entre les WAL archivés et les sauvegardes effectuées (si pg_basebackup ne les sauvegarde pas lui-même avec l'option `-x`).

Restauration :

pg_basebackup n'offre pas d'outil ni d'option pour la restauration.

La copie est directement utilisable, éventuellement après déplacement, décompression des `.tar.gz`. Mais généralement on ajoutera un `recovery.signal` (`recovery.conf` jusqu'en v11), et on définira la `restore_command` pour récupérer les archives. Dans l'idéal, `restore_command` sera déjà prête dans `postgresql.conf` (ou un modèle pour `recovery.conf`).

Si le but est de monter un serveur secondaire de l'instance copiée, il existe une option utile :`--write-recovery-conf` (ou `-R`), qui génère la configuration nécessaire dans le répertoire de la sauvegarde (`recovery.conf`, ou `postgresql.auto.conf` et fichier vide `standby.signal`). avec les paramètres pour une réplication en *streaming*.

1.3 PGBACKREST - PRÉSENTATION GÉNÉRALE

- David Steele (Crunchy Data)
- Langage : **C**
- License : **MIT** (libre)
- Type d'interface : **CLI** (ligne de commande)

1.3.1 PGBACKREST - FONCTIONNALITÉS

- Gère la sauvegarde et la restauration
  - *pull* ou *push*, multidépôts
  - mono ou multi-serveurs
- Indépendant des commandes système
  - utilise un protocole dédié
- Sauvegardes complètes, différentielles ou incrémentales
- Multi-thread, sauvegarde depuis un secondaire, asynchrone...
- Projet récent mature

pgBackRest[5] est un outil de gestion de sauvegardes PITR écrit en perl et en C, par David Steele de Crunchy Data.

Il met l'accent sur les performances avec de gros volumes et les fonctionnalités, au prix d'une complexité à la configuration :

- un protocole dédié pour le transfert et la compression des données ;
- des opérations parallélisables en multi-thread ;
- la possibilité de réaliser des sauvegardes complètes, différentielles et incrémentielles ;
- la possibilité d'archiver ou restaurer les WAL de façon asynchrone, et donc plus rapide ;
- la possibilité d'abandonner l'archivage en cas d'accumulation et de risque de saturation de `pg_wal` ;
- la gestion de dépôts de sauvegarde multiples (pour sécuriser notamment),
- le support intégré de dépôts S3 ou Azure ;
- la sauvegarde depuis un serveur secondaire ;
- le chiffrement des sauvegardes ;
- la restauration en mode delta, très pratique pour restaurer un serveur qui a décroché mais n'a que peu divergé.

Le projet est récent, très actif, considéré comme fiable, et les fonctionnalités proposées sont intéressantes.

Pour la supervision de l'outil, une sonde Nagios est fournie par un des développeurs : check_pgbackrest[6] .

---

[5] https://pgbackrest.org/
[6] https://github.com/pgstef/check_pgbackrest/

1.3.2 PGBACKREST - SAUVEGARDES

- Type de sauvegarde : **physique/PITR** (à chaud)
- Type de stockage : **local**, **push** ou **pull**
- Planification : **crontab**
- Complètes, différentielles et incrémentales
- Compression des WAL

pgBackRest gère uniquement des sauvegardes physiques.

Il peut fonctionner soit en local (directement sur le serveur hébergeant l'instance à sauvegarder) pour un stockage local des sauvegardes, soit être exécuté depuis un serveur distant, déléguant ainsi l'ordonnancement, la compression et le stockage des données à celui-ci.

La technique utilisée pour la prise de sauvegarde repose sur le mécanisme interne standard et historique : `pg_start_backup()`, copie des fichiers, `pg_stop_backup()`.

1.3.3 PGBACKREST - RESTAURATION

- Locale ou à distance
- Point dans le temps : date, identifiant de transaction, timeline ou point de restauration

La restauration d'une sauvegarde peut se faire soit localement, si les sauvegardes sont stockées en local, soit à distance. Dans ce dernier cas, les données à restaurer seront transférées via SSH.

Plusieurs types de point dans le temps peuvent être utilisés comme cible :

- la date ;
- un identifiant de transaction ;
- une timeline (en cas de divergence de timeline, `pgBackRest` peut restaurer les transactions issues d'une timeline précise) ;
- un point de restauration créé par un appel préalable à la fonction :
  - `pg_create_restore_point()`.

1.3.4 PGBACKREST - INSTALLATION

- Accéder au dépôt communautaire PGDG
- Installer le paquet `pgbackrest`

pgBackRest est disponible sur le dépôt communautaire maintenu par la communauté PostgreSQL pour les systèmes d'exploitation disposant des gestionnaires de paquet au format deb (Debian, Ubuntu…)[7] ou rpm (Red Hat, Rocky Linux, CentOS, Fedora…)[8] .

Il est recommandé de manière générale de privilégier une installation à partir de ces paquets plutôt que par les sources, essentiellement pour des raisons de maintenance.

---

[7] https://apt.postgresql.org/pub/repos/apt/
[8] https://yum.postgresql.org/

1.3.5 PGBACKREST - UTILISATION

```
Usage:
    pgbackrest [options] [command]

Commands:
    archive-get     Get a WAL segment from the archive.
    archive-push    Push a WAL segment to the archive.
    backup          Backup a database cluster.
    check           Check the configuration.
    expire          Expire backups that exceed retention.
    help            Get help.
    info            Retrieve information about backups.
    restore         Restore a database cluster.
    stanza-create   Create the required stanza data.
    stanza-delete   Delete a stanza.
    stanza-upgrade  Upgrade a stanza.
    start           Allow pgBackRest processes to run.
    stop            Stop pgBackRest processes from running.
    version         Get version.
```

pgBackRest propose différentes commandes pouvant être passées en argument afin de contrôler les actions.

L'usage de ces différentes commandes sera détaillé ultérieurement.

1.3.6 PGBACKREST - CONFIGURATION

- `/etc/pgbackrest.conf`
- Configuration générale dans la section `[global]`
- Chaque instance à sauvegarder doit avoir sa propre section, appelée `stanza`

Le format de configuration `INI` permet de définir des sections, qui sont matérialisées sous la forme d'une ligne : `[nomdesection]`.

pgBackRest s'attend à lire un fichier de configuration contenant la section `[global]`, contenant les paramètres de configuration globaux, et une section par instance à sauvegarder.

pgBackRest utilise le terme `stanza` pour regrouper l'ensemble des configurations à appliquer pour une instance à sauvegarder.

Exemple de configuration :

```
[global]
repo1-path=/var/lib/pgsql/10/backups
```

```
[ma_stanza]
pg1-path=/var/lib/pgsql/10/data
```

1.3.7 PGBACKREST - CONFIGURATION POSTGRESQL

- Adapter l'archivage dans le fichier `postgresql.conf`

```
archive_mode = on
wal_level = replica
archive_command = 'pgbackrest --stanza=ma_stanza archive-push %p'
```

Il est nécessaire d'activer l'archivage des journaux de transactions en positionnant le paramètre `archive_mode` à `on` et en définissant un niveau d'enregistrement d'informations dans les journaux de transactions (`wal_level`) supérieur ou égal à `replica` (ou `archive` avant la version 9.6).

pgBackRest fournit une commande permettant de simplifier la configuration de l'archivage. Pour l'utiliser, il faut configurer le paramètre `archive_command` pour qu'il utilise l'option `archive-push` de la commande `pgbackrest`. Il faut également fournir à cette commande le nom de la `stanza` à utiliser.

1.3.8 PGBACKREST - CONFIGURATION GLOBALE

- Fichier `pgbackrest.conf`
- Section **[global]** pour la configuration globale

```
[global]
process-max=1
repo1-path=/var/lib/pgbackrest
```

- **process-max** : nombre de processus maximum à utiliser pour la compression et le transfert des sauvegardes ;
- **repo1-path** : chemin où seront stockées les sauvegardes et les archives ;
- **repo-cipher-pass** : passphrase à utiliser pour chiffrer/déchiffrer le répertoire des sauvegardes ;
- **log-level-console** : par défaut à `warn`, définit le niveau de traces des commandes exécutées en console.

1.3.9 PGBACKREST - CONFIGURATION DE LA RÉTENTION

- Type de rétention des sauvegardes complètes

```
repo1-retention-full-type=count|time
```

- **Nombre** de sauvegardes complètes

```
repo1-retention-full=2
```

- **Nombre** de sauvegardes différentielles

```
repo1-retention-diff=3
```

La politique de rétention des sauvegardes complètes peut être configurée avec l'option `repo1-retention-full-type`. Elle peut prendre deux valeurs :

- `count` : le nombre de sauvegardes à conserver, c'est la valeur par défaut ;
- `time` : un nombre de jours pendant lequel on doit pouvoir restaurer, c'est-à-dire que l'on doit avoir au moins une sauvegarde plus vieille que ce nombre de jours.

Voici un exemple pour illustrer la mode de rétention `time`, dont le fonctionnement n'est pas très intuitif. Si l'on dispose des trois sauvegardes complètes suivantes :

- F1 : 25 jours ;
- F2 : 20 jours ;
- F3 : 10 jours.

Avec une rétention de 15 jours, seule la sauvegarde F1 sera supprimée. F2 sera conservée, car il doit exister au moins une sauvegarde de plus de 15 jours pour garantir de pouvoir restaurer pendant cette période.

Il est possible de différencier le nombre de sauvegardes complètes et différentielles. La rétention pour les sauvegardes différentielles ne peut être définie qu'en nombre.

Lorsqu'une sauvegarde complète expire, toutes les sauvegardes différentielles et incrémentales qui lui sont associées expirent également.

1.3.10 PGBACKREST - CONFIGURATION SSH

- Utilisateur `postgres` pour les serveurs PostgreSQL
- Échanger les clés SSH publiques entre les serveurs PostgreSQL et le serveur de sauvegarde
- Configurer `repo1-host*` dans la `pgbackrest.conf`

Dans le cadre de la mise en place de sauvegardes avec un stockage des données sur un serveur tiers, pgBackRest fonctionnera par SSH.

Il est donc impératif d'autoriser l'authentification SSH par clé, et d'échanger les clés publiques entre les différents serveurs hébergeant les instances PostgreSQL et le serveur de sauvegarde.

Il faudra ensuite adapter les paramètres `repo1-host*` dans la configuration de pgBackRest.

- **repo1-host** : hôte à joindre par SSH ;
- **repo1-host-user** : utilisateur pour la connexion SSH ;
- ...

1.3.11 PGBACKREST - CONFIGURATION PAR INSTANCE

- Une section par instance
  - appelée `stanza`

Après avoir vu les options globales, nous allons voir à présent les options spécifiques à chaque instance à sauvegarder.

1.3.12 PGBACKREST - EXEMPLE CONFIGURATION PAR INSTANCE

- Section spécifique par instance
- Permet d'adapter la configuration aux différentes instances
- Exemple

```
[ma_stanza]
pg1-path=/var/lib/pgsql/10/data
```

Une `stanza` définit l'ensemble des configurations de sauvegardes pour un cluster PostgreSQL spécifique. Chaque section `stanza` définit l'emplacement du répertoire de données ainsi que l'hôte/utilisateur si le cluster est distant. Chaque configuration de la partie globale peut être surchargée par `stanza`.

Le nom de la `stanza` est important et doit être significatif car il sera utilisé lors des tâches d'exploitation pour identifier l'instance cible.

Il est également possible d'ajouter ici des `recovery-option` afin de personnaliser les options du fichier `recovery.conf` (ou `postgresql.auto.conf` en version 12 et supérieures) qui sera généré automatiquement à la restauration d'une sauvegarde.

1.3.13 PGBACKREST - INITIALISER LE RÉPERTOIRE DE STOCKAGE DES SAUVEGARDES

- Pour initialiser le répertoire de stockage des sauvegardes

```
$ sudo -u postgres pgbackrest --stanza=ma_stanza stanza-create
```

- Vérifier la configuration de l'archivage

```
$ sudo -u postgres pgbackrest --stanza=ma_stanza check
```

La commande d'initialisation doit être lancée sur le serveur où se situe le répertoire de stockage après que la `stanza` ait été configurée dans `pgbackrest.conf`.

La commande `check` valide que pgBackRest et le paramètre `archive_command` soient correctement configurés. Les commandes `pg_create_restore_point('pgBackRest Archive Check')` et `pg_switch_wal()` sont appelées à cet effet pour forcer PostgreSQL à archiver un segment WAL.

1.3.14 PGBACKREST - EFFECTUER UNE SAUVEGARDE

- Pour déclencher une nouvelle sauvegarde complète

```
$ sudo -u postgres pgbackrest --stanza=ma_stanza --type=full backup
```

- Types supportés : `incr`, `diff`, `full`
- La plupart des paramètres peuvent être surchargés

Exemple de sortie d'une sauvegarde complète :

```
$ sudo -u postgres pgbackrest --stanza=ma_stanza --type=full backup |grep P00
P00   INFO: backup command begin 2.19: --log-level-console=info
--no-log-timestamp --pg1-path=/var/lib/pgsql/12/data --process-max=1
--repo1-path=/var/lib/pgsql/12/backups --repo1-retention-full=1
--stanza=ma_stanza --type=full
P00   INFO: execute non-exclusive pg_start_backup() with label
"pgBackRest backup started at 2019-11-26 12:39:26":
backup begins after the next regular checkpoint completes
P00   INFO: backup start archive = 000000010000000000000005, lsn = 0/5000028
P00   INFO: full backup size = 24.2MB
P00   INFO: execute non-exclusive pg_stop_backup() and wait for all WAL
segments to archive
P00   INFO: backup stop archive = 000000010000000000000005, lsn = 0/5000100
P00   INFO: new backup label = 20191126-123926F
P00   INFO: backup command end: completed successfully
P00   INFO: expire command begin 2.19: --log-level-console=info
--no-log-timestamp --pg1-path=/var/lib/pgsql/12/data --process-max=1
--repo1-path=/var/lib/pgsql/12/backups --repo1-retention-full=1
--stanza=ma_stanza --type=full
```

```
P00   INFO: expire full backup 20191126-123848F
P00   INFO: remove expired backup 20191126-123848F
P00   INFO: expire command end: completed successfully
```

La commande se charge automatiquement de supprimer les sauvegardes devenues obsolètes.

1.3.15 PGBACKREST - LISTER LES SAUVEGARDES

- Lister les sauvegardes présentes et leur taille

```
$ sudo -u postgres pgbackrest --stanza=ma_stanza info
```

Exemple de sortie de la commande :

```
$ sudo -u postgres pgbackrest --stanza=ma_stanza info
stanza: ma_stanza
    status: ok
    cipher: none

    db (current)
        wal archive min/max (12-1): 000000010000000000000005/000000010000000000000005

        full backup: 20191126-123926F
            timestamp start/stop: 2019-11-26 12:39:26 / 2019-11-26 12:39:37
            wal start/stop: 000000010000000000000005 / 000000010000000000000005
            database size: 24.2MB, backup size: 24.2MB
            repository size: 2.9MB, repository backup size: 2.9MB
```

1.3.16 PGBACKREST - PLANIFICATION

- Pas de planificateur intégré
  - le plus simple est d'utiliser `cron`

La planification des sauvegardes peut être faite par n'importe quel outil de planification de tâches, le plus connu étant `cron`.

pgBackRest maintient les traces de ses activités par défaut dans `/var/log/pgbackrest` avec un niveau de traces plus élevé qu'en console. Il n'est donc généralement pas nécessaire de gérer cela au niveau de la planification.

1.3.17 PGBACKREST - DÉPÔTS

- Plusieurs dépôts simultanés possibles
  - Sauvegarde des journaux en parallèle
  - `--repo1-option=…` , appel avec `--repo=1`
- Types : POSIX (NFS, ssh), CIFS
- Cloud : S3, Azure

pgBackRest permet de maintenir plusieurs dépôts de sauvegarde simultanément[9] .

Un intérêt est de gérer des rétentions différentes. Par exemple un dépôt local contiendra juste les dernières sauvegardes et journaux, alors qu'un deuxième dépôt sera sur un autre site plus lointain, éventuellement moins cher, et/ou une rétention supérieure.

Les propriétés des différents dépôts (type, chemin, rétention...) se définissent avec les options `repo1-path`, `repo2-path`, etc. Désigner un dépôt particulier se fait avec `--repo=1` par exemple.

Une sauvegarde se font en désignant le dépôt, mais l'archivage est simultané sur tous les dépôts. L'archivage asynchrone est conseillé dans ce cas.

Les types de dépôts supportés sont ceux montés sur le serveur ou accessibles par ssh, NFS (avec la même attention aux options de montage que pour PostgreSQL[10]), CIFS (avec des restrictions sur les liens symboliques ou le fsync), mais aussi ceux à base de *buckets* : S3 ou compatible, Google Cloud, et Azure Blob.

1.3.18 PGBACKREST - RESTAURATION

- Effectuer une restauration

```
$ sudo -u postgres pgbackrest --stanza=ma_stanza restore
```

- Nombreuses options à la restauration, notamment :
  - `--delta`
  - `--target` / `--type`

Exemple de sortie de la commande :

```
$ sudo -u postgres pgbackrest --stanza=ma_stanza restore |grep P00

P00   INFO: restore command begin 2.19: --log-level-console=info
--no-log-timestamp --pg1-path=/var/lib/pgsql/12/data
--process-max=1 --repo1-path=/var/lib/pgsql/12/backups --stanza=ma_stanza
```

[9] https://pgbackrest.org/configuration.html#section-repository
[10] https://docs.postgresql.fr/current/creating-cluster.html#CREATING-CLUSTER-FILESYSTEM

```
P00   INFO: restore backup set 20191126-123926F
P00   INFO: write updated /var/lib/pgsql/12/data/postgresql.auto.conf
P00   INFO: restore global/pg_control (performed last to ensure aborted
restores cannot be started)
P00   INFO: restore command end: completed successfully
```

L'option `--delta` permet de ne restaurer que les fichiers qui seraient différents entre la sauvegarde et le répertoire de données déjà présent sur le serveur. Elle permet de gagner beaucoup de temps pour reprendre une restauration qui a été interrompue pour une raison ou une autre, pour resynchroniser une instance qui a « décroché », pour restaurer une version légèrement antérieure ou postérieure dans du PITR.

La cible à restaurer peut être spécifiée avec `--target`, associé à `--type`. Par exemple, pour restaurer à une date précise sur une timeline précise :

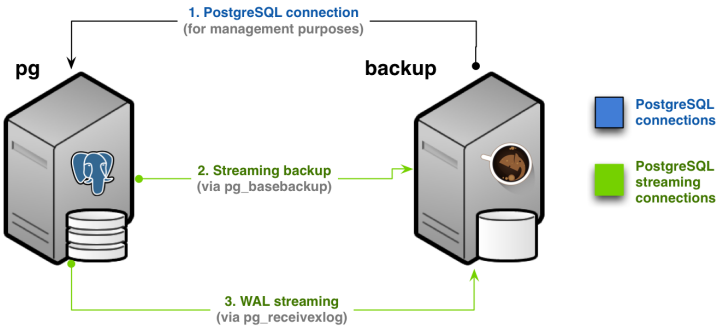
```
pgbackrest --stanza=instance --delta \
  --type=time --target='2020-07-16 11:07:00' \
  --target-timeline=4 \
  --target-action=pause \
  --set=20200716-102845F \
  restore
```

1.4 BARMAN - PRÉSENTATION GÉNÉRALE

- 2ndQuadrant Italia
- Langage: **python** >= 3.4
- OS: **Unix/Linux**
- Versions compatibles: **>= 8.3**
- License: **GPL3** (libre)
- Type d'interface: **CLI** (ligne de commande)

`barman` est un outil développé avec le langage python, compatible uniquement avec les environnements Linux/Unix. Il a été développé par la société 2ndQuadrant Italia (à présent partie de EDB) et distribué sous license GPL3.

1.4.1 BARMAN - SCÉNARIO « STREAMING-ONLY »

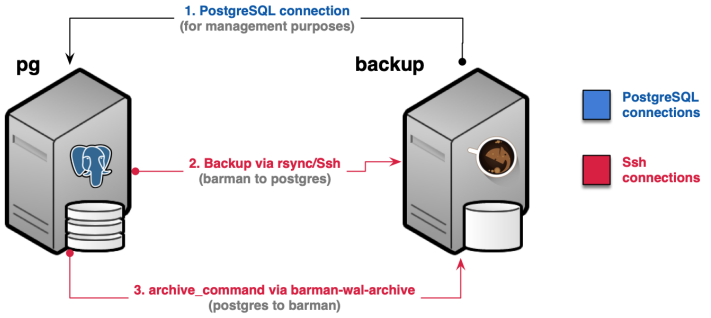


Le scénario évoqué ci-dessus est communément appelé `streaming-only` puisqu'il ne requiert pas de connexion SSH pour les opérations de sauvegardes et d'archivage.

En effet, les outils `pg_basebackup` et `pg_receivewal` sont utilisés pour ces opérations et se basent donc uniquement sur le protocole de réplication.

Attention, rien ne garantit que l'instance sauvegardée conserve bien les journaux WAL nécessaires. Il faudra donc utiliser l'archivage ou un slot de réplication comme filet de sécurité.

1.4.2 BARMAN - SCÉNARIO « RSYNC-OVER-SSH »



Ce deuxième scénario se base donc sur une connexion SSH afin de réaliser les sauvegardes et récupérer les archives des journaux WAL.

1.4.3 BARMAN - SAUVEGARDES

- Type de sauvegarde : **physique/PITR** (à chaud)
- Type de stockage : **local** ou **pull**
- Planification : **crontab**
- Méthodes :
  - **pg_start_backup() / rsync / pg_stop_backup()**
  - **pg_basebackup / pg_receivewal**
- Incrémentales : si **rsync + hardlink**
- Compression des WAL

Barman gère uniquement des sauvegardes physiques.

Il peut fonctionner soit en local (directement sur le serveur hébergeant l'instance à sauvegarder) pour un stockage local des sauvegardes, et peut aussi être exécuté depuis un serveur distant, déléguant ainsi l'ordonnancement, la compression et le stockage des données.

La technique utilisée pour la prise de sauvegarde repose sur le mécanisme interne standard et historique : `pg_start_backup()`, copie des fichiers, `pg_stop_backup()`.

Contrairement aux autres outils présentés, Barman peut également se servir de `pg_basebackup` et `pg_receivewal` pour récupérer les sauvegardes et les archives des journaux WAL.

Il est possible d'activer la dé-duplication de fichiers entre deux sauvegardes lorsque la méthode via `rsync` est employée.

1.4.4 BARMAN - SAUVEGARDES (SUITE)

- Limitation du débit réseau lors des transferts
- Compression des données lors des transferts via le réseau
- Sauvegardes concurrentes
- Hook pre/post sauvegarde
- Hook pre/post archivage WAL
- Compression WAL : `gzip`, `bzip2`, `pigz`, `pbzip2`, etc.
- Pas de compression des données (sauf WAL)

Barman supporte la limitation du débit réseau lors du transfert des données sur un serveur tiers, ainsi que la compression des données à la volée le temps du transfert.

Quatre niveaux de scripts ancrés (*hooks*) sont possibles :

- **avant la sauvegarde** ;
- **après la sauvegarde** ;
- **avant l'archivage d'un WAL** ;
- **après l'archivage d'un WAL**.

Attention, l'opération d'archivage citée ici est celle effectuée par Barman lorsqu'il déplace et compresse un WAL à partir du répertoire `incoming_wals/` vers le répertoire `wals/`, il ne s'agit pas de l'archivage au sens PostgreSQL.

1.4.5 BARMAN - POLITIQUE DE RÉTENTION

- **Durée** (jour/semaine)
- **Nombre** de sauvegardes

La politique de rétention peut être exprimée soit en nombre de sauvegardes à conserver, soit en fenêtre de restauration : une semaine, deux mois, etc.

1.4.6 BARMAN - RESTAURATION

- Locale ou à distance
- Point dans le temps : date, identifiant de transaction, timeline ou point de restauration

La restauration d'une sauvegarde peut se faire soit localement, si les sauvegardes sont stockées en local, soit à distance. Dans ce dernier cas, les données à restaurer seront transférées via SSH.

Plusieurs types de point dans le temps peuvent être utilisés comme cible :

- la date ;
- un identifiant de transaction ;
- une timeline (en cas de divergence de timeline, `barman` peut restaurer les transactions issues d'une timeline précise) ;
- un point de restauration créé par un appel préalable à la fonction :
  - `pg_create_restore_point()`.

1.4.7 BARMAN - INSTALLATION

- Accéder au dépôt communautaire PGDG
- Installer le paquet `barman`

Barman est disponible sur le dépôt communautaire maintenu par la communauté PostgreSQL pour les systèmes d'exploitation disposant des gestionnaires de paquet au format deb (Debian, Ubuntu...)[11] ou rpm (Red Hat, Rocky Linux, CentOS, Fedora...)[12] .

Il est recommandé de manière générale de privilégier une installation à partir des paquets issus du PGDG plutôt que par les sources, essentiellement pour des raisons de maintenance.

[11] https://apt.postgresql.org/pub/repos/apt/
[12] https://yum.postgresql.org/

1.4.8 BARMAN - UTILISATION

```
usage: barman [-h] [-v] [-c CONFIG] [--color {never,always,auto}] [-q] [-d]
              [-f {json,console}]

              {archive-wal,backup,check,check-backup,cron,delete,diagnose,
              get-wal,list-backup,list-files,list-server,put-wal,
              rebuild-xlogdb,receive-wal,recover,show-backup,show-server,
              replication-status,status,switch-wal,switch-xlog,sync-info,
              sync-backup,sync-wals}
[...]
optional arguments:
  -h, --help            show this help message and exit
  -v, --version         show program's version number and exit
  -c CONFIG, --config CONFIG
                        uses a configuration file (defaults: ~/.barman.conf,
                        /etc/barman.conf, /etc/barman/barman.conf)
  --color {never,always,auto}, --colour {never,always,auto}
                        Whether to use colors in the output (default: 'auto')
  -q, --quiet           be quiet (default: False)
  -d, --debug           debug output (default: False)
  -f {json,console}, --format {json,console}
                        output format (default: 'console')
```

Barman propose différentes commandes pouvant être passées en argument afin de contrôler les actions.

L'usage de ces différentes commandes sera détaillé ultérieurement.

L'option `-c` (ou `--config`) permet d'indiquer l'emplacement du fichier de configuration. L'option `-q` (ou `--quiet`) désactive l'envoi de messages sur la sortie standard.

1.4.9 BARMAN - CONFIGURATION

- `/etc/barman.conf`
- Format `INI`
- Configuration générale dans la section `[barman]`
- Chaque instance à sauvegarder doit avoir sa propre section
- Un fichier de configuration par instance via la directive :

```
configuration_files_directory = /etc/barman.d
```

Le format de configuration `INI` permet de définir des sections, qui sont matérialisées sous la forme d'une ligne : `[nomdesection]`.

Barman s'attend à lire un fichier de configuration contenant la section `[barman]`, contenant

les paramètres de configuration globaux, et une section par instance à sauvegarder, le nom de la section définissant ainsi le nom de l'instance.

Pour des questions de lisibilité, il est possible de créer un fichier de configuration par instance à sauvegarder. Ce fichier doit alors se trouver (par défaut) dans le dossier `/etc/barman.d`. Le nom du fichier doit se terminer par `.conf` pour être pris en compte.

1.4.10 BARMAN - CONFIGURATION UTILISATEUR

- Utilisateur système `barman`

L'utilisateur système `barman` est utilisé pour les connexions SSH. Il faut donc penser à générer ses clés RSA, les échanger et établir une première connexion avec les serveurs hébergeant les instances PostgreSQL à sauvegarder.

1.4.11 BARMAN - CONFIGURATION SSH

- Utilisateur `postgres` pour les serveurs PostgreSQL
- Utilisateur `barman` pour le serveur de sauvegardes
- Générer les clés SSH (RSA) des utilisateurs système `postgres` (serveurs PG) et `barman` (serveur barman)
- Échanger les clés SSH publiques entre les serveurs PostgreSQL et le serveur de sauvegarde
- Établir manuellement une première connexion SSH entre chaque machine
- Inutile si utilisation de `pg_basebackup` / `pg_receivewal`

Dans le cadre de la mise en place de sauvegardes avec un stockage des données sur un serveur tiers, la plupart des outils et méthodes historiques de sauvegardes reposent sur le protocole SSH et des outils tels que `rsync` pour assurer les transferts au travers du réseau.

Afin d'automatiser ces transferts via le protocole SSH, il est impératif d'autoriser l'authentification SSH par clé, et d'échanger les clés publiques entre les différents serveurs hébergeant les instances PostgreSQL et le serveur de sauvegarde.

1.4.12 BARMAN - CONFIGURATION POSTGRESQL

- Adapter la configuration de l'archivage dans le fichier `postgresql.conf` :

```
wal_level = 'replica'
archive_mode = on
archive_command = 'barman-wal-archive backup-srv pgsrv %p'
```

- ... ou paramétrer la réplication si utilisation de `pg_basebackup` / `pg_receivewal`

Le paramétrage de l'archivage des journaux de transactions reste classique. La directive `archive_command` doit faire appel directement à l'outil système en charge du transfert du fichier.

Le paramètre `archive_mode` peut prendre la valeur `always` pour permettre un archivage à partir des serveurs secondaires.

Depuis la version 2.6 de Barman, il est recommandé d'utiliser la commande `barman-wal-archive` intégrée (fournie par le paquet `barman-cli`) pour gérer l'archivage. Cette commande interagit directement avec Barman pour recevoir le fichier, écrire son contenu *via* `fsync` et l'envoyer dans le répertoire *incomming* adapté. Cela réduit donc le risque de corruption, perte de données ou simplement d'erreur de répertoire.

1.4.13 BARMAN - CONFIGURATION GLOBALE

- `barman.conf`

```
[barman]
barman_home = /var/lib/barman
barman_user = barman
log_file = /var/log/barman/barman.log
log_level = INFO
configuration_files_directory = /etc/barman.d
```

- **barman_home** : répertoire racine de travail de Barman, contenant les sauvegardes et les journaux de transactions archivés ;
- **barman_user** : utilisateur système ;
- **log_file** : fichier contenant les traces Barman ;
- **configuration_files_directory**: chemin vers le dossier d'inclusion des fichiers de configuration supplémentaires (défaut : `/etc/barman.d`) ;
- **log_level** : niveau de verbosité des traces, par défaut `INFO`.

1.4.14 BARMAN - CONFIGURATION SAUVEGARDES

- Configuration globale des options de sauvegarde

```
compression = gzip
immediate_checkpoint = false
basebackup_retry_times = 0
basebackup_retry_sleep = 30
```

- **compression** : méthode de compression des journaux de transaction - sont disponibles : `gzip`, `bzip2`, `custom`, laissant la possibilité d'utiliser l'utilitaire de compression de son choix (défaut : `gzip`) ;
- **immediate_checkpoint** : force la création immédiate d'un checkpoint impliquant une augmentation des écritures, le but étant de débuter la sauvegarde le plus rapidement possible (defaut : `off`) ;
- **basebackup_retry_times** : nombre de tentative d'écriture d'un fichier - utile pour relancer la copie d'un fichier en cas d'échec sans compromettre le déroulement global de la sauvegarde ;
- **basebackup_retry_sleep** : spécifié en secondes, il s'agit ici de l'intervalle de temps entre deux tentatives de copie d'un fichier en cas d'échec.

1.4.15 BARMAN - CONFIGURATION RÉSEAU

- Possibilité de réduire la bande passante
- Et de compresser le trafic réseau
- Exemple

```
bandwidth_limit = 4000
network_compression = false
```

- **bandwidth_limit** : limitation de l'utilisation de la bande passante réseau lors du transfert de la sauvegarde, s'exprime en `kbps` (par défaut à `0`, autrement dit pas de limitation) ;
- **network_compression** : activation de la compression à la volée des données lors du transfert réseau de la sauvegarde - utilisé à la sauvegarde ou lors d'une restauration (défaut : `false`).

1.4.16 BARMAN - CONFIGURATION RÉTENTION

- Configuration de la rétention en nombre de sauvegardes
- Ou en « fenêtre de restauration », en jours, semaines ou mois
- Déclenchement d'une erreur en cas de sauvegarde trop ancienne
- Exemple

```
minimum_redundancy = 5
retention_policy = RECOVERY WINDOW OF 7 DAYS
last_backup_maximum_age = 2 DAYS
```

- **minimum_redundancy** : nombre minimum de sauvegardes à conserver - si ce n'est pas respecté, Barman empêchera la suppression (défaut : `0`) ;
- **retention_policy** : définit la politique de rétention en s'exprimant soit en nombre de sauvegarde via la syntaxe `REDUNDANCY <valeur>`, soit en fenêtre de restauration via la syntaxe `RECOVERY OF <valeur> {DAYS | WEEKS | MONTHS}` (défaut : aucune rétention appliquée) ;
- **last_backup_maximum_age** : expression sous la forme `<value> {DAYS | WEEKS | MONTHS}`, définit l'âge maximal de la dernière sauvegarde - si celui-ci n'est pas respecté, lors de l'utilisation de la commande `barman check`, une erreur sera levée.

1.4.17 BARMAN - CONFIGURATION DES HOOKS

- Lancer des scripts avant ou après les sauvegardes
- Et avant ou après le traitement du WAL archivé par Barman
- Exemple :

```
pre_backup_script = ...
post_backup_script = ...
pre_archive_script = ...
post_archive_script = ...
```

Barman offre la possibilité d'exécuter des commandes externes (scripts) avant et/ou après les opérations de sauvegarde et les opérations d'archivage des journaux de transaction.

Attention, la notion d'archivage de journal de transactions dans ce contexte ne concerne pas l'archivage réalisé depuis l'instance PostgreSQL, qui copie les WAL dans un répertoire `<incoming>` sur le serveur Barman, mais bien l'opération de récupération du WAL depuis ce répertoire `<incoming>`.

1.4.18 BARMAN - CONFIGURATION PAR INSTANCE

- `configuration_files_directory`
  - un fichier de configuration par instance
- Ou une section par instance

Après avoir vu les options globales, nous allons voir à présent les options spécifiques à chaque instance à sauvegarder.

Afin de conserver une certaine souplesse dans la gestion de la configuration Barman, il est recommandé de paramétrer la directive `configuration_files_directory` de la section `[barman]` afin de pouvoir charger d'autres fichiers de configuration, permettant ainsi d'isoler la section spécifique à chaque instance à sauvegarder dans son propre fichier de configuration.

1.4.19 BARMAN - EXEMPLE CONFIGURATION PAR INSTANCE

- Section spécifique par instance
- Permet d'adapter la configuration aux différentes instances
- Exemple

```
[pgsrv]
description = "PostgreSQL Instance pgsrv"
ssh_command = ssh postgres@pgsrv
conninfo = host=pgsrv user=postgres dbname=postgres
backup_method = rsync
reuse_backup = link
backup_options = exclusive_backup
archiver = on
```

La première ligne définit le nom de la section. Ce nom est important et doit être significatif car il sera utilisé lors des tâches d'exploitation pour identifier l'instance cible.

L'idéal est d'utiliser le nom d'hôte ou l'adresse IP du serveur si celui-ci n'héberge qu'une seule instance.

- **description** : chaîne de caractère servant de descriptif de l'instance ;
- **ssh_command** : commande shell utilisée pour établir la connexion `ssh` vers le serveur hébergeant l'instance à sauvegarder ;
- **conninfo** : chaîne de connexion PostgreSQL.

Tous les autres paramètres, à l'exception de `log_file` et `log_level`, peuvent être redéfinis pour chaque instance.

1.4.20 BARMAN - EXEMPLE CONFIGURATION STREAMING ONLY

```
[pgsrv]
description =  "Sauvegarde de pgsrv via Streaming Replication"
conninfo = host=pgsrv user=barman dbname=postgres
streaming_conninfo = host=pgsrv user=streaming_barman
backup_method = postgres
streaming_archiver = on
slot_name = barman
```

1.4.21 BARMAN - VÉRIFICATION DE LA CONFIGURATION

- La commande `show-server` montre la configuration

```
$ sudo -u barman barman show-server {<instance> | all}
```

- La commande `check` effectue des tests pour la valider

```
$ sudo -u barman barman check {<instance> | all}
```

La commande `show-server` permet de visualiser la configuration de Barman pour l'instance spécifiée, ou pour toutes les instances si le mot-clé `all` est utilisé.

C'est particulièrement utile lors d'un premier paramétrage, notamment pour récupérer la valeur assignée au paramètre `incoming_wals_directory` afin de configurer correctement la valeur du paramètre `archive_command` de l'instance à sauvegarder.

La commande `check` vérifie le bon paramétrage de Barman pour l'instance spécifiée, ou pour toutes les instances si le mot-clé `all` est utilisé.

Elle permet de s'assurer que les points clés sont fonctionnels, tels que l'accès SSH, l'archivage des journaux de transaction (`archive_command`, `archive_mode`...), la politique de rétention, la compression, etc.

Il est possible d'utiliser l'option `--nagios` qui permet de formater la sortie de la commande `check` et de l'utiliser en tant que sonde Nagios.

Exemple de sortie de la commande `show-server` :

```
$ barman show-server pgsrv
Server pgsrv:
  active: True
  archive_command: None
  archive_mode: None
  archiver: True
  archiver_batch_size: 0
  backup_directory: /var/lib/barman/pgsrv
  backup_method: rsync
```

```
backup_options: BackupOptions(['exclusive_backup'])
bandwidth_limit: None
barman_home: /var/lib/barman
barman_lock_directory: /var/lib/barman
basebackup_retry_sleep: 30
basebackup_retry_times: 0
basebackups_directory: /var/lib/barman/pgsrv/base
check_timeout: 30
compression: None
conninfo: host=pgsrv user=postgres dbname=postgres
create_slot: manual
current_xlog: None
custom_compression_filter: None
custom_decompression_filter: None
data_directory: None
description: PostgreSQL Instance pgsrv
disabled: False
errors_directory: /var/lib/barman/pgsrv/errors
immediate_checkpoint: False
incoming_wals_directory: /var/lib/barman/pgsrv/incoming
is_in_recovery: None
is_superuser: None
last_backup_maximum_age: None
max_incoming_wals_queue: None
minimum_redundancy: 0
msg_list: []
name: pgsrv
network_compression: False
parallel_jobs: 1
passive_node: False
path_prefix: None
pgespresso_installed: None
post_archive_retry_script: None
post_archive_script: None
post_backup_retry_script: None
post_backup_script: None
post_delete_retry_script: None
post_delete_script: None
post_recovery_retry_script: None
post_recovery_script: None
post_wal_delete_retry_script: None
post_wal_delete_script: None
postgres_systemid: None
pre_archive_retry_script: None
pre_archive_script: None
pre_backup_retry_script: None
```

```
  pre_backup_script: None
  pre_delete_retry_script: None
  pre_delete_script: None
  pre_recovery_retry_script: None
  pre_recovery_script: None
  pre_wal_delete_retry_script: None
  pre_wal_delete_script: None
  primary_ssh_command: None
  recovery_options: RecoveryOptions([])
  replication_slot: None
  replication_slot_support: None
  retention_policy: None
  retention_policy_mode: auto
  reuse_backup: link
  server_txt_version: None
  slot_name: None
  ssh_command: ssh postgres@pgsrv
  streaming_archiver: False
  streaming_archiver_batch_size: 0
  streaming_archiver_name: barman_receive_wal
  streaming_backup_name: barman_streaming_backup
  streaming_conninfo: host=pgsrv user=postgres dbname=postgres
  streaming_wals_directory: /var/lib/barman/pgsrv/streaming
  synchronous_standby_names: None
  tablespace_bandwidth_limit: None
  wal_retention_policy: main
  wals_directory: /var/lib/barman/pgsrv/wals
```

Exemple de sortie de la commande `check` :

```
$ barman check pgsrv
Server pgsrv:
  PostgreSQL: OK
  is_superuser: OK
  wal_level: OK
  directories: OK
  retention policy settings: OK
  backup maximum age: OK (no last_backup_maximum_age provided)
  compression settings: OK
  failed backups: OK (there are 0 failed backups)
  minimum redundancy requirements: OK (have 0 backups, expected at least 0)
  ssh: OK (PostgreSQL server)
  not in recovery: OK
  systemid coherence: OK (no system Id stored on disk)
  archive_mode: OK
  archive_command: OK
  continuous archiving: OK
```

```
  archiver errors: OK
```

1.4.22 BARMAN - STATUT

- La commande `status` affiche des informations détaillées
  - sur la configuration Barman
  - sur l'instance spécifiée
- Exemple

```
$ sudo -u barman barman status {<instance> | all}
```

La commande `status` retourne de manière détaillée le statut de l'instance spécifiée, ou de toutes si le mot-clé `all` est utilisé.

Les informations renvoyées sont, entre autres :

- la description extraite du fichier de configuration de Barman ;
- la version de PostgreSQL ;
- si l'extension `pgespresso` est utilisée ;
- l'emplacement des données sur l'instance (`PGDATA`) ;
- la valeur de l'`archive_command` ;
- des informations sur les journaux de transactions :
  - position courante
  - dernier segment archivé
- des informations sur les sauvegardes :
  - nombre de sauvegarde
  - ID de la première sauvegarde
  - ID de la dernière sauvegarde
  - politique de rétention

Exemple de sortie de la commande :

```
$ barman status pgsrv
Server pgsrv:
  Description: PostgreSQL Instance pgsrv
  Active: True
  Disabled: False
  PostgreSQL version: 12.1
  Cluster state: in production
  pgespresso extension: Not available
  Current data size: 24.4 MiB
  PostgreSQL Data directory: /var/lib/pgsql/12/data
  Current WAL segment: 000000010000000000000004
  PostgreSQL 'archive_command' setting: barman-wal-archive localhost pgsrv %p
```

```
Last archived WAL: 000000010000000000000003, at Wed Dec 11 11:44:12 2019
Failures of WAL archiver: 52 (000000010000000000000001 at Wed Dec 11 11:44:04 2019)
Server WAL archiving rate: 1.41/hour
Passive node: False
Retention policies: not enforced
No. of available backups: 0
First available backup: None
Last available backup: None
Minimum redundancy requirements: satisfied (0/0)
```

1.4.23 BARMAN - DIAGNOSTIQUER

- La commande `diagnose` renvoie
  - les informations renvoyées par la commande `status`
  - des informations supplémentaires (sur le système par exemple)
  - au format `json`
- Exemple

```
$ sudo -u barman barman diagnose
```

La commande `diagnose` retourne les informations importantes concernant toutes les instances à sauvegarder, en donnant par exemple les versions de chacun des composants utilisés.

Elle reprend également les informations retournées par la commande `status`, le tout au format JSON.

1.4.24 BARMAN - NOUVELLE SAUVEGARDE

- Pour déclencher une nouvelle sauvegarde

```
$ sudo -u barman barman backup {<instance> | all}
```

- Le détail de sauvegarde effectuée est affiché en sortie

La commande `backup` lance immédiatement une nouvelle sauvegarde, pour une seule instance si un identifiant est passé en argument, ou pour toutes les instances configurées si le mot-clé `all` est utilisé.

Exemple de sortie de la commande :

```
$ barman backup pgsrv
Starting backup using rsync-exclusive method for server pgsrv in
                                /var/lib/barman/pgsrv/base/20191211T121244
Backup start at LSN: 0/5000028 (000000010000000000000005, 00000028)
```

```
This is the first backup for server pgsrv
WAL segments preceding the current backup have been found:
  000000010000000000000001 from server pgsrv has been removed
  000000010000000000000002 from server pgsrv has been removed
  000000010000000000000003 from server pgsrv has been removed
Starting backup copy via rsync/SSH for 20191211T121244
Copy done (time: 1 second)
This is the first backup for server pgsrv
Asking PostgreSQL server to finalize the backup.
Backup size: 24.3 MiB. Actual size on disk: 24.3 MiB (-0.00% deduplication ratio).
Backup end at LSN: 0/5000138 (000000010000000000000005, 00000138)
Backup completed (start time: 2019-12-11 12:12:44.788598, elapsed time: 5 seconds)
Processing xlog segments from file archival for pgsrv
  000000010000000000000004
  000000010000000000000005
  000000010000000000000005.00000028.backup
```

1.4.25 BARMAN - LISTER LES SAUVEGARDES

- Pour lister les sauvegardes existantes

```
$ sudo -u barman barman list-backup {<instance> | all}
```

- Affiche notamment la taille de la sauvegarde et des WAL associés

Liste les sauvegardes du catalogue, soit par instance, soit toutes si le mot-clé `all` est passé en argument.

Exemple de sortie de la commande :

```
$ barman list-backup pgsrv
pgsrv 20191211T121244 - Wed Dec 11 12:12:47 2019 - Size: 40.3 MiB -
                        WAL Size: 0 B
```

1.4.26 BARMAN - DÉTAIL D'UNE SAUVEGARDE

- `show-backup` affiche le détail d'une sauvegarde (taille...)

```
$ sudo -u barman barman show-backup <instance> <ID-sauvegarde>
```

- `list-files` affiche le détail des fichiers d'une sauvegarde

```
$ sudo -u barman barman list-files <instance> <ID-sauvegarde>
```

La commande `show-backup` affiche toutes les informations relatives à une sauvegarde en particulier, comme l'espace disque occupé, le nombre de journaux de transactions associés, etc.

La commande `list-files` permet quant à elle d'afficher la liste complète des fichiers contenus dans la sauvegarde.

Exemple de sortie de la commande `show-backup` :

```
$ barman show-backup pgsrv 20191211T121244
Backup 20191211T121244:
  Server Name            : pgsrv
  System Id              : 6769104211696624889
  Status                 : DONE
  PostgreSQL Version     : 120001
  PGDATA directory       : /var/lib/pgsql/12/data

  Base backup information:
    Disk usage           : 24.3 MiB (40.3 MiB with WALs)
    Incremental size     : 24.3 MiB (-0.00%)
    Timeline             : 1
    Begin WAL            : 000000010000000000000005
    End WAL              : 000000010000000000000005
    WAL number           : 1
    Begin time           : 2019-12-11 12:12:44.526305+01:00
    End time             : 2019-12-11 12:12:47.794687+01:00
    Copy time            : 1 second + 1 second startup
    Estimated throughput : 14.3 MiB/s
    Begin Offset         : 40
    End Offset           : 312
    Begin LSN           : 0/5000028
    End LSN             : 0/5000138

  WAL information:
    No of files          : 0
    Disk usage           : 0 B
    Last available       : 000000010000000000000005

  Catalog information:
    Retention Policy     : not enforced
    Previous Backup      : - (this is the oldest base backup)
    Next Backup          : - (this is the latest base backup)
```

1.4.27 BARMAN - SUPPRESSION D'UNE SAUVEGARDE

- Pour supprimer manuellement une sauvegarde

```
$ sudo -u barman barman delete <instance> <ID-sauvegarde>
```

- Renvoie une erreur si la redondance minimale ne le permet pas

La suppression d'une sauvegarde nécessite de spécifier l'instance ciblée et l'identifiant de la sauvegarde à supprimer.

Cet identifiant peut être trouvé en utilisant la commande Barman `list-backup`.

Si le nombre de sauvegardes (après suppression) ne devait pas respecter le seuil défini par la directive `minimum_redundancy`, la suppression ne sera alors pas possible.

1.4.28 BARMAN - TÂCHES DE MAINTENANCE

- La commande Barman `cron` déclenche la maintenance
  - récupération des WAL archivés
  - compression
  - politique de rétention
- Exemple

```
$ sudo -u barman barman cron
```

- À planifier !

La commande `cron` permet d'exécuter les tâches de maintenance qui doivent être exécutées périodiquement, telles que l'archivage des journaux de transactions (déplacement du dossier `incoming_wals/` vers `wals/`), ou la compression.

L'application de la politique de rétention est également faite dans ce cadre.

Le démarrage de la commande `pg_recievewal` est aussi gérée par ce biais.

L'exécution de cette commande doit donc être planifiée via votre ordonnanceur préféré (`cron` d'Unix par exemple), par exemple toutes les minutes.

Si vous avez installé Barman via les paquets (rpm ou debian), une tâche `cron` exécutée toutes les minutes a été créée automatiquement.

1.4.29 BARMAN - RESTAURATION

- Copie/transfert de la sauvegarde
- Copie/transfert des journaux de transactions
- Génération du fichier `recovery.conf`
- Copie/transfert des fichiers de configuration

Le processus de restauration géré par Barman reste classique, mais nécessite tout de même quelques points d'attention.

En particulier, les fichiers de configuration sauvegardés sont restaurés dans le dossier `$PGDATA`, or ce n'est potentiellement pas le bon emplacement selon le type d'installation / configuration de l'instance. Dans une installation basée sur les paquets Debian/Ubuntu par exemple, les fichiers de configuration se trouvent dans `/etc/postgresql/<version>/<instance>` et non dans le répertoire PGDATA. Il convient donc de penser à les supprimer du `PGDATA` s'ils n'ont rien à y faire avant de démarrer l'instance.

De même, la directive de configuration `archive_command` est passée à `false` par Barman. Une fois l'instance démarrée et fonctionnelle, il convient de modifier la valeur de ce paramètre pour réactiver l'archivage des journaux de transactions.

1.4.30 BARMAN - OPTIONS DE RESTAURATION

- Locale ou à distance
- Cibles : timeline, date, ID de transaction ou point de restauration
- Déplacement des tablespaces

Au niveau de la restauration, Barman offre la possibilité de restaurer soit en local (sur le serveur où se trouvent les sauvegardes), soit à distance.

Le cas le plus commun est une restauration à distance, car les sauvegardes sont généralement centralisées sur le serveur de sauvegarde d'où Barman est exécuté.

Pour la restauration à distance, Barman s'appuie sur la couche SSH pour le transfert des données.

Barman supporte différents types de cibles dans le temps pour la restauration :

- **timeline** : via l'option `--target-tli`, lorsqu'une divergence de timeline a eu lieu, il est possible de restaurer et rejouer toutes les transactions d'une timeline particulière ;

- **date** : via l'option `--target-time` au format **YYYY-MM-DD HH:MM:SS.mmm**, spécifie une date limite précise dans le temps au delà de laquelle la procédure de restauration arrête de rejouer les transactions ;
- **identifiant de transaction** : via l'option `--target-xid`, restauration jusqu'à une transaction précise ;
- **point de restauration** : via l'option `--target-name`, restauration jusqu'à un point de restauration créé préalablement sur l'instance via l'appel à la fonction `pg_create_restore_point(nom)`.

Barman permet également de relocaliser un tablespace lors de la restauration.

Ceci est utile lorsque l'on souhaite restaurer une sauvegarde sur un serveur différent, ne disposant pas des même points de montage des volumes que l'instance originelle.

1.4.31 BARMAN - EXEMPLE DE RESTAURATION À DISTANCE

- Exemple d'une restauration
  - déclenchée depuis le serveur Barman
  - avec un point dans le temps spécifié

```
$ sudo -u barman barman recover                  \
    --remote-ssh-command "ssh postgres@pgsrv"    \
    --target-time "2019-12-11 14:00:00"          \
    pgsrv 20191211T121244 /var/lib/pgsql/12/data/
```

Dans cet exemple, nous souhaitons effectuer une restauration à distance via l'option `--remote-ssh-command`, prenant en argument `"ssh postgres@pgsrv"` correspondant à la commande SSH pour se connecter au serveur à restaurer.

L'option `--target-time` définit ici le point de restauration dans le temps comme étant la date « 2019-12-11 14:00:00 ».

Les trois derniers arguments sont :

- l'identifiant de l'instance dans le fichier de configuration de Barman : `pgsrv` ;
- l'identifiant de la sauvegarde cible : `20191211T121244` ;
- et enfin le dossier PGDATA de l'instance à restaurer.

1.5 PITRERY - PRÉSENTATION GÉNÉRALE

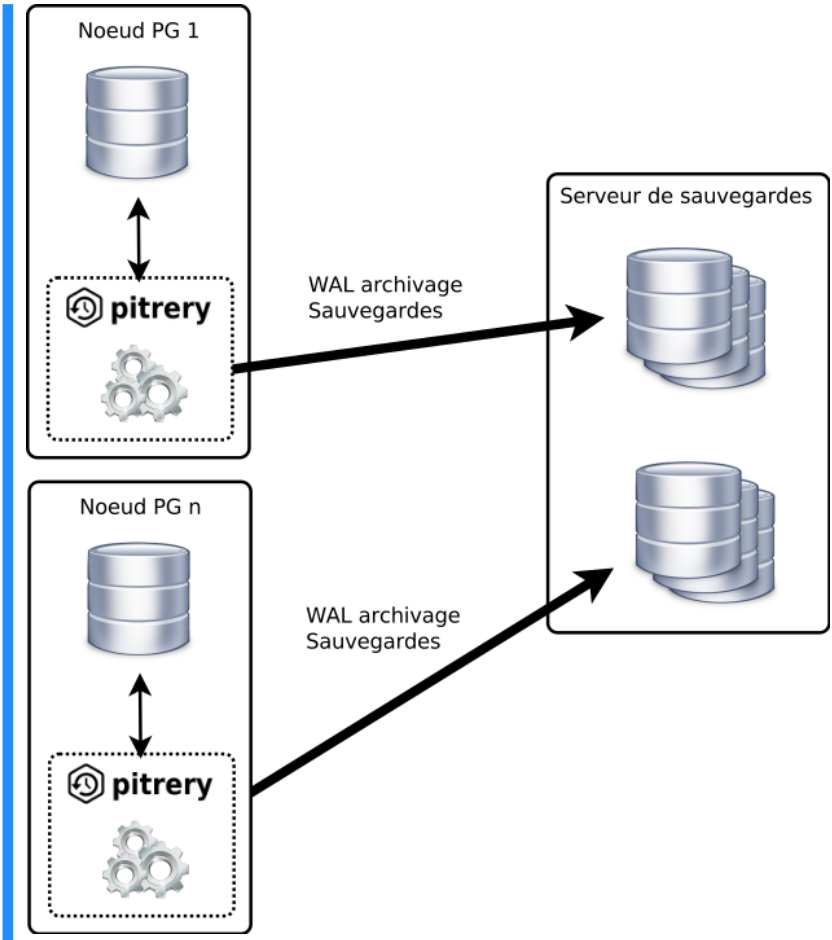
- R&D Dalibo
- Langage : **bash**
- OS : **Unix/Linux**
- Versions compatibles : **8.2 à 14**
- License : **BSD** (libre)
- Type d'interface : **CLI** (ligne de commande)
- Développement arrêté, mode LTS

pitrery est un outil de gestion de sauvegarde physique et restauration PITR, écrit en bash, issu du labo R&D de Dalibo.

Il est compatible avec tous les environnements Unix/Linux disposant de l'interpréteur de shell `bash`, et supporte toutes les versions de PostgreSQL depuis la 8.2 jusqu'à la version 14, qui sera la dernière version supportée.

En effet, le support de l'outil s'arrêtera en novembre 2026.

1.5.1 PITRERY - DIAGRAMME



1.5.2 PITRERY - SAUVEGARDES

- Type de sauvegarde : **physique/PITR**
- Type de stockage : **local** ou **distant** (push) via `rsync/ssh`
- Planification : **crontab**
- Méthodes : **pg_start_backup() / rsync ou tar / pg_stop_backup()**
- Compression : **gzip**, **bzip2**, **pigz**, **pbzip2**, etc.
- Compression des WAL

- Scripts pre/post sauvegarde (hooks)
- Déduplication de fichier (`rsync` + liens physiques *hardlinks*)

pitrery permet de gérer exclusivement des sauvegardes physiques.

Il s'installe sur le même serveur que l'instance à sauvegarder, et est capable de stocker les sauvegardes sur un point de montage local ou de les pousser au travers d'une connexion SSH vers un serveur de sauvegarde tiers.

La méthode de sauvegarde interne utilisée se base sur les appels de fonction SQL `pg_start_backup()` et `pg_stop_backup()`, les données étant effectivement copiées entre ces deux appels.

L'archivage des journaux de transactions doit être activé, et peut se faire en appelant un script interne à `pitrery`.

Les sauvegardes et les journaux de transactions peuvent être compressés, avec possibilité de spécifier l'outil de compression à utiliser.

pitrery propose également deux *hooks* (points d'ancrage) dans le code, offrant ainsi la possibilité d'exécuter des scripts externes avant ou après l'exécution d'une nouvelle sauvegarde.

Concernant le stockage des sauvegardes, il est possible d'activer la déduplication des fichiers entre deux sauvegardes.

Ceci est possible en utilisant `rsync` comme format de sauvegarde. Dans ce cas, pitrery va tirer parti de la capacité de `rsync` à effectuer une sauvegarde différentielle combinée avec la création de liens physiques (*hardlink*), ce qui permet selon les cas d'économiser de l'espace disque et du temps de sauvegarde de manière conséquente.

La compression de la sauvegarde est également possible si l'on a choisi `tar` comme format de sauvegarde.

1.5.3 PITRERY - POLITIQUE DE RÉTENTION

- **Durée** (en jours)
- **Nombre** de sauvegardes

La politique de rétention des sauvegardes se définit en nombre de jours ou de sauvegardes.

La combinaison des deux paramétrages est possible.

Dans ce cas, pitrery déclenchera la suppression d'une sauvegarde **si et seulement si** les deux rétentions sont dépassées.

Ainsi, si la durée spécifiée est de sept jours, et le nombre de sauvegardes est de trois, alors, au moment de supprimer une sauvegarde antérieure à sept jours, pitrery vérifiera qu'il reste bien au moins trois sauvegardes disponibles.

1.5.4 PITRERY - RESTAURATION

- À partir des données locales ou distantes
- Point dans le temps : date

Comme dans le cadre de la sauvegarde, lors de la restauration, pitrery est capable de transférer les données depuis un serveur tiers.

Il est également possible de spécifier un point dans le temps pour effectuer une restauration de type PITR.

1.5.5 PITRERY - INSTALLATION

- Téléchargement depuis le site du projet
  - installation depuis les sources (tarball)
  - paquets RPM et DEB disponibles

Pitrery est disponible sur le dépôt communautaire maintenu par la communauté PostgreSQL pour les systèmes d'exploitation disposant des gestionnaires de paquet au format rpm (Red Hat, Rocky Linux, CentOS, Fedora...)[13] .

Les sources ainsi qu'un paquet DEB sont également disponibles dans la page de téléchargement[14] .

[13] https://yum.postgresql.org/
[14] https://dalibo.github.io/pitrery/downloads.html

Il est recommandé de manière générale de privilégier une installation à partir de paquets plutôt que par les sources, essentiellement pour des raisons de maintenance.

Pour installer depuis les sources, il faut d'abord télécharger la dernière version stable depuis le site du projet, désarchiver le fichier, puis installer `pitrery` via la commande `make install`, à exécuter avec les droits root.

En cas d'installation via les sources, les scripts pitrery sont installés dans le dossier `/usr/local/bin`, et le fichier de configuration se trouve dans le dossier `/usr/local/etc/pitrery`. En cas d'utilisation avec les paquets, les répertoires suivants sont utilisés : `/usr/bin` et `/etc/pitrery`.

Exemple détaillé :

```
wget https://github.com/dalibo/pitrery/releases/download/v3.3/pitrery-3.3.tar.gz
tar xvzf pitrery-3.3.tar.gz
cd pitrery-3.3
sudo make install
```

1.5.6 PITRERY - UTILISATION

```
usage: pitrery [options] action [args]

options:
    -f file      Path to the configuration file
    -l           List configuration files in the default directory
    -V           Display the version and exit
    -?           Print help

actions:
    list - Display information about backups
    backup - Perform a base backup
    restore - Restore a base backup and prepare PITR
    purge - Clean old base backups and archived WAL files
    check - Verify configuration and backups integrity
    configure - Create a configuration file from the command line
    help - Print help, optionnally for an action
```

L'option `-f` permet de définir l'emplacement du fichier de configuration. Par défaut, celui-ci se trouve ici sous `/usr/local/etc/pitrery/pitrery.conf` ou `/etc/pitrery/pitrery.conf` suivant le type d'installation.

L'option `-V` affiche la version de l'outil et se termine.

L'option `-?` permet d'afficher le message d'aide, et peut être combinée avec une commande pour afficher l'aide spécifique à la commande (par exemple, `pitrery restore -?`).

Les actions possibles sont les suivantes :

- **list** pour lister le contenu du catalogue de sauvegarde ;
- **backup** pour effectuer une sauvegarde ;
- **restore** pour effectuer une restauration ;
- **purge** pour supprimer les sauvegardes et journaux de transaction ne satisfaisant plus la politique de rétention ;
- **check** pour remonter l'état des sauvegardes et des archives à une sonde nagios en fonction de la rétention souhaitée ;
- **configure** pour construire le fichier de configuration en ligne de commande.

1.5.7 PITRERY - CONFIGURATION POSTGRESQL

- Adapter l'archivage dans le fichier `postgresql.conf`

```
archive_mode = on
wal_level = replica
archive_command = 'archive_wal %p'
```

(NB : il pourrait être prudent de renseigner le chemin complet d'`archive_wal`.)

Il est tout d'abord nécessaire d'activer l'archivage des journaux de transactions en positionnant le paramètre `archive_mode` à `on` et en définissant un niveau d'enregistrement d'informations dans les journaux de transactions (`wal_level`) supérieur ou égal à `replica` (ou `archive` avant la version 9.6).

Il est à noter que le paramètre `archive_mode` peut prendre la valeur `always` à partir de la version 9.5 pour permettre un archivage à partir des serveurs secondaires.

pitrery fournit un script permettant de simplifier la configuration de la commande d'archivage. Pour l'utiliser, il faut configurer le paramètre `archive_command` pour qu'il appelle le script `archive_wal`, suivi de l'argument `%p` qui correspond au dossier contenant les journaux de transactions de l'instance.

Remarque : avant la version 3.0 de pitrery, le script `archive_wal` se nommait `archive_xlog`.

1.5.8 PITRERY - CONFIGURATION SSH

- Stockage des sauvegardes sur un serveur distant
- Générer les clés SSH (RSA) des utilisateurs système `postgres` sur chacun des serveurs (nœud PG et serveur de sauvegarde)
- Échanger les clés SSH publiques entre les serveurs PostgreSQL et le serveur de sauvegarde
- Établir manuellement une première connexion SSH entre chaque serveur

Dans le cadre de la mise en place de sauvegardes avec un stockage des données sur un serveur tiers, la plupart des outils et méthodes historiques de sauvegardes se reposent sur le protocole SSH et les outils tels que `rsync` pour assurer les transferts au travers du réseau.

Afin d'automatiser ces transferts reposant sur le protocole SSH, il est impératif d'autoriser l'authentification SSH par clé et d'échanger les clés publiques des différents serveurs.

1.5.9 PITRERY - FICHIER DE CONFIGURATION

- Emplacement par défaut :
  - `/usr/local/etc/pitrery/pitrery.conf` (installation par les souces)
  - `/etc/pitrery/pitrery.conf` (installation par paquet)
- Possibilité de spécifier un autre emplacement

Il est possible d'utiliser un fichier de configuration spécifique avec l'option `-c <fichier>` lors de l'appel de la commande `pitrery`.

1.5.10 PITRERY - CONFIGURATION CONNEXION POSTGRESQL

- Options de connexion à l'instance

```
PGPSQL="psql"
PGUSER="postgres"
PGPORT=5432
PGHOST="/tmp"
PGDATABASE="postgres"
```

Ces paramètres sont utilisés par pitrery pour établir une connexion à l'instance afin d'exécuter les fonctions `pg_start_backup()` et `pg_stop_backup()` utilisées pour signaler à PostgreSQL qu'une sauvegarde PITR démarre et se termine.

- **PGPSQL** : chemin vers le binaire du client `psql` ;
- **PGUSER** : superutilisateur PostgreSQL de connexion ;
- **PGPORT** : numéro de port d'écoute PostgreSQL ;
- **PGHOST** : hôte PostgreSQL (dossier contenant le socket UNIX de connexion ou adresse IP ou alias) ;
- **PGDATABASE** : nom de la base de données de connexion.

1.5.11 PITRERY - LOCALISATION

- Configurer les emplacements
- Pour l'instance à sauvegarder

```
PGDATA="/var/lib/pgsql/data"
PGWAL=
```

- Pour la destination des sauvegardes

```
BACKUP_DIR="/var/lib/pgsql/backups/pitr"
```

- Pour la destination des WAL archivés

```
ARCHIVE_DIR="$BACKUP_DIR/archived_wal"
```

- **PGDATA** : répertoire contenant les données de l'instance PostgreSQL à sauvegarder ;
- **PGWAL** : répertoire contenant les journaux de transactions de PostgreSQL, à laisser vide s'il se trouve à la racine du `PGDATA` ;
- **BACKUP_DIR** : répertoire destiné à contenir l'arborescence des sauvegardes PITR ;
- **ARCHIVE_DIR** : répertoire destiné à contenir les journaux de transactions archivés.

Il est possible d'utiliser des paramètres initialisés précédemment comme variables lors de l'initialisation des paramètres suivants.

1.5.12 PITRERY - CONFIGURATION DU MODE DE SAUVEGARDE

- Paramètre de configuration `STORAGE`
- Deux modes possibles :
  - `tar` : sauvegarde complète, éventuellement compressée
  - `rsync` : sauvegarde complète ou différentielle, pas de compression

Le paramètre `STORAGE` permet de spécifier deux modes de sauvegarde.

Le premier, `tar`, réalise la sauvegarde sous la forme d'une archive `tar` du répertoire `PGDATA`, éventuellement compressée, plus autant d'archives `tar` qu'il y a de tablespaces dans l'instance. Ce mode ne permet que les sauvegardes complètes.

Le second, `rsync`, indique à pitrery de copier les fichiers de l'instance à l'aide de la commande système idoine. Les fichiers dans le répertoire de la sauvegarde seront donc exactement les mêmes que ceux de l'instance :

- le contenu du répertoire `PGDATA` de l'instance est dans un sous-répertoire `pgdata` ;
- le contenu des différents tablespaces est dans un sous-répertoire `tblspc/<nom du tblspc>`.

Ce mode ne permet pas la compression, il consommera donc par défaut beaucoup plus de place que la sauvegarde `tar`. En revanche, ce mode permet la réalisation de sauvegarde différentielles, en réalisant des liens physiques (*hardlink*) des fichiers depuis une précédente sauvegarde et en utilisant la fonctionnalité de `rsync` basée sur les checksums pour ne sauvegarder que les fichiers ayant été modifiés. Si l'instance a eu peu de modifications depuis la précédente sauvegarde, ce mode peut donc faire économiser du temps de sauvegarde et de l'espace disque.

1.5.13 PITRERY - CONFIGURATION DE LA RÉTENTION

- Configuration de la rétention en nombre de sauvegardes

```
PURGE_KEEP_COUNT=3
```

- Ou en jours

```
PURGE_OLDER_THAN=7
```

- Les deux paramètres peuvent être combinés

Le paramétrage de la rétention peut s'effectuer à deux niveaux :

- **PURGE_KEEP_COUNT** : nombre minimal de sauvegarde à conserver ;
- **PURGE_OLDER_THAN** : âge minimal des sauvegardes à conserver, exprimé en jours.

Si les deux paramètres sont utilisés de concert, une sauvegarde ne sera supprimée que si elle ne répond à aucune des deux rétentions.

Ainsi, avec la configuration faite ici, une sauvegarde datant de plus de sept jours ne sera supprimée que s'il reste effectivement au moins trois autres sauvegardes.

1.5.14 PITRERY - CONFIGURATION DE L'ARCHIVAGE

- L'archivage peut être configuré pour :
  - archiver en local (montage NFS...)
  - archiver vers un serveur distant en SSH
- Exemple

```
ARCHIVE_LOCAL="no"
ARCHIVE_HOST=bkpsrv
ARCHIVE_USER=pitrery
ARCHIVE_COMPRESS="yes"
```

- Utilisé par la commande `archive_wal`

Ce paramétrage est utilisé lors de l'appel au script `archive_wal` par l'instance PostgreSQL :

- **ARCHIVE_LOCAL** : `yes` ou `no`, définit si les journaux sont archivés localement ou sur un serveur de sauvegarde distant (rsync/ssh) ;
- **ARCHIVE_HOST** : hôte stockant les journaux de transactions archivés, si `ARCHIVE_LOCAL = no` ;
- **ARCHIVE_USER** : utilisateur SSH de connexion vers le serveur distant pour le transfert des WAL, si `ARCHIVE_LOCAL = no` ;
- **ARCHIVE_COMPRESS** : compression des journaux de transactions lors de l'archivage.

1.5.15 PITRERY - CONFIGURATION DE LA COMPRESSION

- Configuration de la compression
  - des WAL archivés par la commande `archive_wal`
  - des sauvegardes effectuées au format `tar`
- Exemple

```
COMPRESS_BIN=
COMPRESS_SUFFIX=
UNCOMPRESS_BIN=
BACKUP_COMPRESS_BIN=
BACKUP_COMPRESS_SUFFIX=
BACKUP_UNCOMPRESS_BIN=
```

Les paramètres indiqués ici permettent de spécifier des commandes spécifiques pour compresser / décompresser les journaux de transactions archivés et les sauvegardes :

- **COMPRESS_BIN** : chemin vers le binaire (+ ses options) utilisé pour la compression des fichiers WAL archivés (`gzip`, `bzip2`, etc), `gzip -4` par défaut ;
- **COMPRESS_SUFFIX** : l'extension du fichier contenant les fichiers WAL archivés compressés, `gz` par défaut, utilisé pour la décompression ;
- **UNCOMPRESS_BIN** : chemin vers l'outil utilisé pour décompresser les fichiers WAL archivés, `gunzip` par défaut ;
- **BACKUP_COMPRESS_BIN** : chemin vers le binaire (+ ses options) utilisé pour la compression des sauvegardes (`gzip`, `bzip2` etc), `gzip -4` par défaut ;
- **BACKUP_COMPRESS_SUFFIX** : l'extension du fichier compressé de la sauvegarde ;
- **BACKUP_UNCOMPRESS_BIN** : chemin vers l'outil utilisé pour décompresser les sauvegardes, `gunzip` par défaut.

Il est possible d'utiliser des outils de compression multithread/multiprocess comme `pigz` ou `pbzip2`.

À noter que la compression de la sauvegarde n'est possible que si la méthode de stockage `tar` est utilisée.

1.5.16 PITRERY - CONFIGURATION DES TRACES

- Activer l'horodatage des traces

```
LOG_TIMESTAMP="yes"
```

- Utiliser syslog pour les traces de l'archivage (`archive_wal`)

```
SYSLOG="no"
SYSLOG_FACILITY="local0"
SYSLOG_IDENT="postgres"
```

Ces paramètres permettent d'activer la redirection des traces vers `syslog` pour les opérations d'archivage et de restauration des journaux de transactions.

Les autres opérations sont écrites vers les sorties standards.

1.5.17 PITRERY - CONFIGURATION DE HOOKS

- Exécuter un script avant ou après une sauvegarde

```
PRE_BACKUP_COMMAND=
POST_BACKUP_COMMAND=
```

Ces paramètres permettent de spécifier des commandes à exécuter avant ou après une sauvegarde :

- **PRE_BACKUP_COMMAND** : exécutée avant le début de la sauvegarde ;
- **POST_BACKUP_COMMAND** : exécutée après la fin de la sauvegarde.

1.5.18 PITRERY - EFFECTUER UNE SAUVEGARDE

- Pour déclencher une nouvelle sauvegarde

```
$ sudo -u postgres pitrery backup
```

- La plupart des paramètres peuvent être surchargés
- Par exemple, l'option `-s` permet de spécifier un mode, `tar` ou `rsync`

L'exécution de la commande pitrery `backup` déclenche la création immédiate d'une nouvelle sauvegarde de l'instance.

Toutes les commandes `pitrery` doivent être exécutées depuis un utilisateur système ayant un accès en lecture aux fichiers de données de l'instance PostgreSQL.

En général, on utilisera le compte de service PostgreSQL, par défaut `postgres`.

Exemple de sortie de la commande pour une sauvegarde `rsync` :

```
$ pitrery backup
INFO: preparing directories in bkpsrv:/var/lib/pitrery/backups/pitr
INFO: listing tablespaces
INFO: starting the backup process
INFO: backing up PGDATA with rsync
INFO: preparing hardlinks from previous backup
INFO: transfering data from /var/lib/pgsql/9.4/data
INFO: backing up tablespace "tmptbs" with rsync
INFO: preparing hardlinks from previous backup
INFO: transfering data from /tmp/tmptbs
INFO: stopping the backup process
NOTICE:  pg_stop_backup complete, all required WAL segments have been archived
INFO: copying the backup history file
INFO: copying the tablespaces list
INFO: backup directory is
      bkpsrv:/var/lib/pitrery/backups/pitr/2015.10.29_22.08.11
INFO: done
```

On voit à la ligne suivante qu'il s'agit d'une sauvegarde différentielle, seuls les fichiers ayant été modifiés depuis la sauvegarde précédente ont réellement été transférés :

```
INFO: preparing hardlinks from previous backup
```

1.5.19 PITRERY - SUPPRESSION DES SAUVEGARDES OBSOLÈTES

- Suppression des sauvegardes et archives ne satisfaisant plus la politique de sauvegarde

```
$ sudo -u postgres pitrery purge
```

- À déclencher systématiquement après une sauvegarde

La commande `purge` se charge d'appliquer la politique de rétention et de supprimer les sauvegardes les plus anciennes, ainsi que les journaux de transactions devenus obsolètes.

1.5.20 PITRERY - LISTER LES SAUVEGARDES

- Lister les sauvegardes présentes et leur taille

```
$ sudo -u postgres pitrery list
```

- L'option `-v` permet d'avoir plus de détails
  - mode de sauvegarde
  - date minimale utilisable pour la restauration
  - taille de chaque tablespace

La commande pitrery `list` énumère les sauvegardes présentes, indiquant l'emplacement, la taille, la date et l'heure de la création de chacune d'elles.

L'option `-v` permet d'afficher des informations plus détaillées, notamment la date minimale utilisable pour une restauration effectuée à partir de cette sauvegarde, ce qui est très pratique pour savoir précisément jusqu'à quel point dans le temps il est possible de remonter en utilisant les sauvegardes présentes.

Exemple de sortie de la commande :

```
$ pitrery list
List of backups on bkpsrv
/var/lib/pitrery/backups/pitr/2015.10.29_22.09.52  36M   2015-10-29 22:09:52 CET
/var/lib/pitrery/backups/pitr/2015.10.29_22.17.15  240M  2015-10-29 22:17:15 CET
/var/lib/pitrery/backups/pitr/2015.10.29_22.28.48  240M  2015-10-29 22:28:48 CET
```

Avec l'option `-v` :

```
$ pitrery list -v
List of backups on bkpsrv
--------------------------------------------------------------------------
Directory:
  /var/lib/pitrery/backups/pitr/2015.10.29_22.09.52
  space used: 36M
  storage: tar with gz compression
Minimum recovery target time:
  2015-10-29 22:09:52 CET
PGDATA:
  pg_default 202 MB
  pg_global 469 kB
Tablespaces:
  "tmptbs" /tmp/tmptbs (24583) 36 MB

--------------------------------------------------------------------------
Directory:
  /var/lib/pitrery/backups/pitr/2015.10.29_22.17.15
  space used: 240M
  storage: rsync
Minimum recovery target time:
  2015-10-29 22:17:15 CET
PGDATA:
  pg_default 202 MB
  pg_global 469 kB
Tablespaces:
  "tmptbs" /tmp/tmptbs (24583) 36 MB

--------------------------------------------------------------------------
```

```
Directory:
  /var/lib/pitrery/backups/pitr/2015.10.29_22.28.48
  space used: 240M
  storage: rsync
Minimum recovery target time:
  2015-10-29 22:28:48 CET
PGDATA:
  pg_default 202 MB
  pg_global 469 kB
Tablespaces:
  "tmptbs" /tmp/tmptbs (24583) 36 MB
```

1.5.21 PITRERY - PLANIFICATION

- Pas de planificateur intégré
  - le plus simple est d'utiliser `cron`
- Exemple

```
00 00 * * * (   pitrery backup                  \
             && pitrery purge)                  \
  >> /var/log/postgresql/pitrery-$(date +\%Y-\%m-\%d).log 2>&1
```

La planification des sauvegardes et de la suppression des sauvegardes ne respectant plus la politique de rétention peut être faite par n'importe quel outil de planification de tâches, le plus connu étant `cron`.

L'exemple montre la planification d'une sauvegarde suivie de la suppression des sauvegardes obsolètes, via la crontab de l'utilisateur système `postgres`.

La sauvegarde est effectuée tous les jours à minuit, et est suivie d'une purge seulement si elle a réussi.

Les traces (stdout et stderr) sont redirigées vers un fichier journalisé par jour.

1.5.22 PITRERY - RESTAURATION

- Effectuer une restauration

```
$ sudo -u postgres pitrery restore
```

- Nombreuses options à la restauration, notamment
  - l'option `-D` permet de modifier la cible du PGDATA
  - l'option `-t` permet de modifier la cible d'un tablespace
  - l'option `-x` permet de modifier la cible du `pg_wal`
  - l'option `-d` permet de spécifier une date de restauration

La restauration d'une sauvegarde se fait par l'appel à la commande pitrery `restore`.

Plusieurs options peuvent être spécifiées, offrant notamment la possibilité de restaurer le PGDATA (option `-D`), les tablespaces (option `-t`) et le pg_wal (option `-x`) dans des répertoires spécifiques.

Il est également possible de restaurer à une date précise (option `-d`), à condition que la date spécifiée soit postérieure à la date minimale utilisable de la plus ancienne sauvegarde disponible.

Voici un exemple combinant ces différentes options :

```
$ pitrery restore -d '2015-10-29 22:10:00' \
                  -D /var/lib/pgsql/9.4/data2/ \
                  -x /var/lib/pgsql/9.4/pg_wal2/ \
                  -t tmptbs:/tmp/tmptbs2/
INFO: searching backup directory
INFO: searching for tablespaces information
INFO:
INFO: backup directory:
INFO:   /var/lib/pitrery/backups/pitr/2015.10.29_22.28.48
INFO:
INFO: destinations directories:
INFO:   PGDATA -> /var/lib/pgsql/9.4/data2/
INFO:   PGDATA/pg_wal -> /var/lib/pgsql/9.4/pg_wal2/
INFO:   tablespace "tmptbs" (24583) -> /tmp/tmptbs2/ (relocated: yes)
INFO:
INFO: recovery configuration:
INFO:   target owner of the restored files: postgres
INFO:   restore_command = '/usr/bin/restore_xlog -h bkpsrv -u pitrery \
                           -d /var/lib/pitrery/archived_wal %f %p'
INFO:   recovery_target_time = '2015-10-29 22:10:00'
INFO:
INFO: checking if /var/lib/pgsql/9.4/data2/ is empty
INFO: setting permissions of /var/lib/pgsql/9.4/data2/
INFO: checking if /var/lib/pgsql/9.4/pg_wal2/ is empty
```

```
INFO: setting permissions of /var/lib/pgsql/9.4/pg_wal2/
INFO: checking if /tmp/tmptbs2/ is empty
INFO: setting permissions of /tmp/tmptbs2/
INFO: transfering PGDATA to /var/lib/pgsql/9.4/data2/ with rsync
INFO: transfer of PGDATA successful
INFO: transfering PGDATA to /var/lib/pgsql/9.4/data2/ with rsync
INFO: transfer of tablespace "tmptbs" successful
INFO: creating symbolic link pg_wal to /var/lib/pgsql/9.4/pg_wal2/
INFO: preparing pg_xlog directory
INFO: preparing recovery.conf file
INFO: done
INFO:
INFO: please check directories and recovery.conf before starting the cluster
INFO: and do not forget to update the configuration of pitrery if needed
INFO:
```

Par cette commande, nous demandons à pitrery de :

- relocaliser le PGDATA dans `/var/lib/pgsql/9.4/data2/` ;
- relocaliser le pg_wal dans `/var/lib/pgsql/9.4/pg_wal2/` ;
- relocaliser le tablespace `tmptbs` dans `/tmp/tmpbbs2/` ;
- inscrire dans le `recovery.conf` que la restauration ne devra pas inclure les transactions au delà du 29 octobre 2015 à 22h10.

pitrery se charge de trouver automatiquement la sauvegarde la plus à jour permettant la restauration de l'instance à cette date précise.

1.6 AUTRES OUTILS DE L'ÉCOSYSTÈME

- De nombreux autres outils existent
  - …ou ont existé
- WAL-E, OmniPITR, pg_rman, walmgr, pg_rman…
- WAL-G

Du fait du dynamisme du projet, l'écosystème des outils autour de PostgreSQL est très changeant.

À côté des trois outils détaillés ci-dessus, que nous recommandons, on trouve de nombreux projets autour du thème de la gestion des sauvegardes.

Certains de ces projets répondent à des problématiques spécifiques, d'autres sont assez anciens et plus guère maintenus (comme WAL-E[15]), rendus inutiles par l'évolution de

[15] https://github.com/wal-e/wal-e

PostgreSQL ces dernières années (comme walmgr, de la suite Skytools[16] , ou OmniPITR[17]) ou simplement peu actifs et peu rencontrés en production (par exemple pg_rman[18] , développé par NTT).

Le plus intéressant et actif est sans doute WAL-G.

1.6.1 WAL-G - PRÉSENTATION

- Successeur de WAL-E, par Citus Data & Yandex
- Orientation cloud
- Aussi pour MySQL et SQL Server

WAL-G[19] est une réécriture d'un ancien outil assez populaire, WAL-E, par Citus[20] et Yandex, et actif.

De par sa conception, il est optimisé pour l'archivage des journaux de transactions vers des stockages *cloud* (Amazon S3, Google, Yandex), la compression multi-processeurs par différents algorithmes[21] et l'optimisation du temps de restauration. Il supporte aussi MySQL et SQL Server (et d'autres dans le futur).

1.7 CONCLUSION

- Des outils pour vous aider !
- Pratiquer, pratiquer et pratiquer
- Superviser les sauvegardes !

Nous venons de vous présenter des outils qui vont vous permettre de vous simplifier la tâche dans la mise en place d'une solution de sauvegarde fiable et robuste de vos instance PostgreSQL.

Cependant, leur maîtrise passera par de la pratique, et en particulier, la pratique de la restauration.

Le jour où la restauration d'une instance de production se présente, ce n'est généralement pas une situation confortable à cause du stress lié à une perte/corruption de données,

[16] https://wiki.postgresql.org/wiki/SkyTools
[17] https://github.com/omniti-labs/omnipitr
[18] https://github.com/ossc-db/pg_rman
[19] https://github.com/wal-g/wal-g
[20] https://www.citusdata.com/blog/2017/08/18/introducing-wal-g-faster-restores-for-postgres/
[21] https://wal-g.readthedocs.io/

interruption du service, etc. Autant maîtriser les outils qui vous permettront de sortir de ce mauvais pas.

N'oubliez pas également l'importance de la supervision des sauvegardes !

1.8 QUIZ

https://dali.bo/i4_quiz

1.9 TRAVAUX PRATIQUES

1.9.1 UTILISATION DE PGBACKREST (OPTIONNEL)

Installer pgBackRest à partir des paquets du PGDG.

En vous aidant de https://pgbackrest.org/user-guide.html#quickstart, configurer pgBackRest pour sauvegarder le serveur PostgreSQL en local dans `/var/lib/pgsql/14/backups`.
Ne conserver qu'une seule sauvegarde complète.

Configurer l'archivage des journaux de transactions de PostgreSQL avec pgBackRest.

Initialiser le répertoire de stockage des sauvegardes et vérifier la configuration de l'archivage.

Lancer une sauvegarde complète. Afficher les détails de cette sauvegarde.

Ajouter des données :
Ajouter une table avec 1 million de lignes.
Forcer la rotation du journal de transaction courant afin de s'assurer que les dernières modifications sont archivées.
Vérifier que le journal concerné est bien dans les archives.

Simulation d'un incident : supprimer tout le contenu de la table.

Restaurer les données avant l'incident à l'aide de pgBackRest.

1.9.2 UTILISATION DE BARMAN (OPTIONNEL)

Installer barman depuis les dépôts communautaires (la documentation est sur https://www.pgbarman.org/documentation/).

Configurer barman pour la sauvegarde du serveur via Streaming Replication (`pg_basebackup` et `pg_receivewal`).

Vérifier que l'archivage fonctionne et que la configuration de barman est correcte.

Faire une sauvegarde.

Ajouter des données :
Ajouter une table avec 1 million de lignes.
Forcer la rotation du journal de transaction courant afin de s'assurer que les dernières modifications sont archivées.

Vérifier que le journal concerné est bien dans les archives.

Lister les sauvegardes.

Afficher les informations sur la sauvegarde.

Simulation d'un incident : supprimer tout le contenu de la table.

Restaurer les données avant l'incident à l'aide de barman.

1.9.3 UTILISATION DE PITRERY (OPTIONNEL)

> Installer pitrery.

> Configurer pitrery pour la sauvegarde du serveur local.

> Faire une sauvegarde.

> Lister les sauvegardes.

> Faire une restauration.

1.10 TRAVAUX PRATIQUES (SOLUTIONS)

1.10.1 UTILISATION DE PGBACKREST (OPTIONNEL)

> Installer pgBackRest à partir des paquets du PGDG.

L'installation du paquet est triviale :

```
# yum install pgbackrest    # CentOS 7
# dnf install pgbackrest    # Rocky Linux
```

> En vous aidant de https://pgbackrest.org/user-guide.html#quickstart, configurer pgBackRest pour sauvegarder le serveur PostgreSQL en local dans `/var/lib/pgsql/14/backups`.
> Ne conserver qu'une seule sauvegarde complète.

Le ficher de configuration est `/etc/pgbackrest.conf` :

```
[global]
repo1-path=/var/lib/pgsql/14/backups
repo1-retention-full=1

[ma_stanza]
pg1-path=/var/lib/pgsql/14/data
```

> Configurer l'archivage des journaux de transactions de PostgreSQL avec pgBackRest.

```
wal_level = replica
archive_mode = on
archive_command = 'pgbackrest --stanza=ma_stanza archive-push %p'
```

Redémarrer PostgreSQL.

> Initialiser le répertoire de stockage des sauvegardes et vérifier la configuration de l'archivage.

Sous l'utilisateur **postgres** :

```
$ pgbackrest --stanza=ma_stanza --log-level-console=info stanza-create
```

Vérifier la configuration de pgBackRest et de l'archivage :

```
$ pgbackrest --stanza=ma_stanza --log-level-console=info check
```

Vérifier que l'archivage fonctionne :

```
$ ls /var/lib/pgsql/14/backups/archive/ma_stanza/14-1/0000000100000000/
```

```
SELECT * FROM pg_stat_archiver;
```

> Lancer une sauvegarde complète. Afficher les détails de cette sauvegarde.

```
$ pgbackrest --stanza=ma_stanza --type=full \
               --log-level-console=info backup |grep P00
```

```
P00   INFO: backup command begin 2.19: --log-level-console=info
--pg1-path=/var/lib/pgsql/14/data --repo1-path=/var/lib/pgsql/14/backups
--repo1-retention-full=1 --stanza=ma_stanza --type=full
P00   INFO: execute non-exclusive pg_start_backup() with label
"pgBackRest backup started at 2021-11-26 12:25:32":
backup begins after the next regular checkpoint completes
P00   INFO: backup start archive = 000000010000000000000003, lsn = 0/3000060
2P00   INFO: full backup size = 24.2MB
P00   INFO: execute non-exclusive pg_stop_backup() and wait for all WAL segments
                                                                    to archive
P00   INFO: backup stop archive = 000000010000000000000003, lsn = 0/3000138
P00   INFO: new backup label = 20211126-122532F
P00   INFO: backup command end: completed successfully (8694ms)
P00   INFO: expire command begin 2.19: --log-level-console=info
--pg1-path=/var/lib/pgsql/14/data --repo1-path=/var/lib/pgsql/14/backups
--repo1-retention-full=1 --stanza=ma_stanza --type=full
P00   INFO: expire command end: completed successfully (8ms)
```

Lister les sauvegardes :

```
$ pgbackrest --stanza=ma_stanza info
stanza: ma_stanza
    status: ok
    cipher: none

    db (current)
        wal archive min/max (14-1): 000000010000000000000003/000000010000000000000003

        full backup: 20211126-122532F
            timestamp start/stop: 2021-11-26 12:25:32 / 2021-11-26 12:25:41
            wal start/stop: 000000010000000000000003 / 000000010000000000000003
            database size: 24.2MB, backup size: 24.2MB
            repository size: 2.9MB, repository backup size: 2.9MB
```

> Ajouter des données :
> Ajouter une table avec 1 million de lignes.
> Forcer la rotation du journal de transaction courant afin de s'assurer que les dernières modifications sont archivées.
> Vérifier que le journal concerné est bien dans les archives.

```
CREATE TABLE matable AS SELECT i FROM generate_series(1,1000000) i ;
SELECT 1000000
```

Forcer la rotation du journal :

```
SELECT pg_switch_wal();
```

Vérifier que le journal concerné est bien dans les archives.

> Simulation d'un incident : supprimer tout le contenu de la table.

```
TRUNCATE TABLE matable;
```

> Restaurer les données avant l'incident à l'aide de pgBackRest.

D'abord, stopper PostgreSQL.

Lancer la commande de restauration :

```
$ pgbackrest --stanza=ma_stanza --log-level-console=info \
--delta \
--target="2021-11-26 12:30:15"  \
--target-action=promote         \
--type=time                     \
--target-exclusive              \
restore |grep P00

P00   INFO: restore command begin 2.19: --delta --log-level-console=info
--pg1-path=/var/lib/pgsql/14/data --repo1-path=/var/lib/pgsql/14/backups
--stanza=ma_stanza --target="2021-11-26 12:30:15" --target-action=promote
--target-exclusive --type=time
P00   INFO: restore backup set 20211126-122532F
P00   INFO: remove invalid files/links/paths from '/var/lib/pgsql/14/data'
P00   INFO: write updated /var/lib/pgsql/14/data/postgresql.auto.conf
P00   INFO: restore global/pg_control
(performed last to ensure aborted restores cannot be started)
P00   INFO: restore command end: completed successfully (501ms)
```

Démarrer PostgreSQL.

Vérifier les logs et la présence de la table disparue.

```
SELECT count(*) FROM matable ;

  count
---------
 1000000
```

Remarque :

Sans spécifier de `--target-action=promote`, on obtiendrait dans les traces de PostgreSQL, après restore :

```
LOG:  recovery has paused
HINT:  Execute pg_wal_replay_resume() to continue.
```

1.10.2 UTILISATION DE BARMAN (OPTIONNEL)

> Installer barman depuis les dépôts communautaires (la documentation est sur https://www.pgbarman.org/documentation/).

Pré-requis : sous CentOS 7, le dépôt EPEL est nécessaire à cause des dépendances python, s'il n'est pas déjà installé :

```
# yum install epel-release
```

La commande suivante suffit pour installer l'outil et ses dépendances.

```
# yum install barman      # CentOS 7
# dnf install barman      # Rocky Linux 8
```

Le paquet crée un utilisateur `barman` qui exécutera la sauvegarde et sera leur propriétaire. L'outil `barman` sera à exécuter uniquement avec cet utilisateur.

> Configurer barman pour la sauvegarde du serveur via Streaming Replication (`pg_basebackup` et `pg_receivewal`).

`/etc/barman.conf` doit contenir :

```
[barman]
barman_user = barman
configuration_files_directory = /etc/barman.d
barman_home = /var/lib/barman
log_file = /var/log/barman/barman.log
log_level = INFO
compression = gzip
```

```
immediate_checkpoint = true
path_prefix = "/usr/pgsql-14/bin"
```

Ce fichier indique que l'utilisateur système est l'utilisateur `barman`. Les sauvegardes et journaux de transactions archivés seront placés dans `/var/lib/barman`.

Puis, il faut créer un fichier par hôte (uniquement `localhost` ici) et le placer dans le répertoire pointé par la variable `configuration_files_directory` (`/etc/barman.d` ici). On y indiquera les chaînes de connexion PostgreSQL pour la maintenance ainsi que pour la réplication.

Dans `/etc/barman.d/`, créez un fichier nommé `localhost.conf` contenant ceci (vous pouvez repartir d'un modèle existant dans ce répertoire) :

```
[localhost]
description =  "Sauvegarde de localhost via Streaming Replication"
conninfo = host=localhost user=barman dbname=postgres
streaming_conninfo = host=localhost user=streaming_barman
backup_method = postgres
streaming_archiver = on
slot_name = barman
```

Il faut donc d'abord créer les utilisateurs qui serviront aux connections :

```
postgres$ createuser --superuser --pwprompt barman
postgres$ createuser --replication --pwprompt streaming_barman
```

Ensuite, il faut s'assurer que ces utilisateurs puissent se connecter sur l'instance PostgreSQL, en modifiant`pg_hba.conf` et peut-être `postgresql.conf`.

```
local   all           barman                          md5
host    all           barman            127.0.0.1/32  md5
host    all           barman            ::1/128       md5
local   replication   streaming_barman                md5
host    replication   streaming_barman  127.0.0.1/32  md5
host    replication   streaming_barman  ::1/128       md5
```

Recharger la configuration (voire redémarrer PostgreSQL si nécessaire).

Configurer les droits du fichier `~/.pgpass` de l'utilisateur système **barman** et ses droits d'accès comme suit :

```
barman$ chmod 600 ~/.pgpass
barman$ cat ~/.pgpass

*:*:*:barman:barmanpwd
*:*:*:streaming_barman:barmanpwd
```

Vérifier maintenant que les utilisateurs peuvent bien se connecter :

```
barman$ psql -c 'SELECT version()' -U barman -h localhost postgres
                                                  version
-------------------------------------------------------------------------
 PostgreSQL 14.1 on x86_64-pc-linux-gnu, compiled by gcc (GCC) 4.8.5 ...

barman$ psql -U streaming_barman -h localhost -c "IDENTIFY_SYSTEM" replication=1
      systemid       | timeline |  xlogpos  | dbname
---------------------+----------+-----------+--------
 6769169214324921667 |        1 | 0/169E438 |
```

Afin d'éviter que le serveur principal ne recycle les journaux que nous souhaitons archiver via le protocole de réplication (et `pg_receivewal`), créer le slot de réplication mentionné dans le fichier de configuration `localhost.conf` :

```
barman$ barman receive-wal --create-slot localhost
Creating physical replication slot 'barman' on server 'localhost'
Replication slot 'barman' created
```

> Vérifier que l'archivage fonctionne et que la configuration de barman est correcte.

Après 1 minute (laissant à la tâche cron le soin de démarrer les processus adéquats), vérifier que l'archivage fonctionne :

```
$ ps -ef |grep streaming_barman
barman   10248 10244  0 14:55 ?        00:00:00 /usr/pgsql-14/bin/pg_receivewal
    --dbname=dbname=replication host=localhost
    options=-cdatestyle=iso replication=true user=streaming_barman
    application_name=barman_receive_wal
    --verbose --no-loop --no-password
    --directory=/var/lib/barman/localhost/streaming --slot=barman

postgres 10249  9575  0 14:55 ?        00:00:00 postgres: walsender
    streaming_barman ::1(49182) streaming 0/169E438
```

On constate bien ici les 2 processus `pg_receivewal` ainsi que `walsender`.

On peut également forcer la génération d'une nouvelle archive :

```
barman$ barman switch-wal localhost --force --archive
The WAL file 000000010000000000000001 has been closed on server 'localhost'
Waiting for the WAL file 000000010000000000000001 from server 'localhost'
Processing xlog segments from streaming for localhost
  000000010000000000000001
```

Vérifier que la configuration de barman est correcte avec la commande suivante :

```
barman$ barman check localhost

Server localhost:
  PostgreSQL: OK
  is_superuser: OK
  PostgreSQL streaming: OK
  wal_level: OK
  replication slot: OK
  directories: OK
  retention policy settings: OK
  backup maximum age: OK (no last_backup_maximum_age provided)
  compression settings: OK
  failed backups: OK (there are 0 failed backups)
  minimum redundancy requirements: OK (have 0 backups, expected at least 0)
  pg_basebackup: OK
  pg_basebackup compatible: OK
  pg_basebackup supports tablespaces mapping: OK
  systemid coherence: OK (no system Id stored on disk)
  pg_receivexlog: OK
  pg_receivexlog compatible: OK
  receive-wal running: OK
  archiver errors: OK
```

> Faire une sauvegarde.

```
barman$ barman backup localhost --wait

Starting backup using postgres method for server localhost in
                                /var/lib/barman/localhost/base/20211111T153507
Backup start at LSN: 0/40000C8 (000000010000000000000004, 000000C8)
Starting backup copy via pg_basebackup for 20211111T153507
Copy done (time: 1 second)
Finalising the backup.
This is the first backup for server localhost
WAL segments preceding the current backup have been found:
  000000010000000000000003 from server localhost has been removed
Backup size: 24.2 MiB
Backup end at LSN: 0/6000000 (000000010000000000000005, 00000000)
Backup completed (start time: 2021-11-11 15:35:07.610047, elapsed time: 2 seconds)
Waiting for the WAL file 000000010000000000000005 from server 'localhost'
Processing xlog segments from streaming for localhost
  000000010000000000000004
Processing xlog segments from streaming for localhost
  000000010000000000000005
```

> Ajouter des données :
> Ajouter une table avec 1 million de lignes.
> Forcer la rotation du journal de transaction courant afin de s'assurer que les dernières modifications sont archivées.

```
CREATE TABLE matable AS SELECT i FROM generate_series(1,1000000) i;
```

Forcer la rotation du journal :

```
SELECT pg_switch_wal();
```

> Vérifier que le journal concerné est bien dans les archives.

Le processus `pg_receivewal` récupère en flux continu les journaux de transactions de l'instance principale dans un fichier `.partial`, présent dans le répertoire `<barman_home>/<instance>/streaming`.

Lors d'une rotation de journal, le fichier est déplacé de façon asynchrone dans le répertoire correspondant au segment auquel il appartient.

```
barman$ find /var/lib/barman/localhost/{streaming,wals} -type f

/var/lib/barman/localhost/streaming/000000010000000000000000A.partial
/var/lib/barman/localhost/wals/xlog.db
/var/lib/barman/localhost/wals/0000000100000000/000000010000000000000003
/var/lib/barman/localhost/wals/0000000100000000/000000010000000000000004
/var/lib/barman/localhost/wals/0000000100000000/000000010000000000000005
/var/lib/barman/localhost/wals/0000000100000000/000000010000000000000006
/var/lib/barman/localhost/wals/0000000100000000/000000010000000000000007
/var/lib/barman/localhost/wals/0000000100000000/000000010000000000000008
/var/lib/barman/localhost/wals/0000000100000000/000000010000000000000009
```

> Lister les sauvegardes.

```
barman$ barman list-backup localhost

localhost 20211111T153507 - Wed Nov 11 15:35:09 2021 - Size: 24.2 MiB -
             WAL Size: 12.5 MiB
```

> Afficher les informations sur la sauvegarde.

```
barman$ barman show-backup localhost 20211111T153507

Backup 20211111T153507:
```

```
  Server Name            : localhost
  System Id              : 6769169214324921667
  Status                 : DONE
  PostgreSQL Version     : 140001
  PGDATA directory       : /var/lib/pgsql/14/data

  Base backup information:
    Disk usage           : 24.2 MiB (24.2 MiB with WALs)
    Incremental size     : 24.2 MiB (-0.00%)
    Timeline             : 1
    Begin WAL            : 000000010000000000000005
    End WAL              : 000000010000000000000005
    WAL number           : 1
    WAL compression ratio: 99.90%
    Begin time           : 2021-11-11 15:35:08+01:00
    End time             : 2021-11-11 15:35:09.509201+01:00
    Copy time            : 1 second
    Estimated throughput : 12.8 MiB/s
    Begin Offset         : 40
    End Offset           : 0
    Begin LSN           : 0/5000028
    End LSN             : 0/6000000

  WAL information:
    No of files          : 4
    Disk usage           : 12.5 MiB
    WAL rate             : 266.82/hour
    Compression ratio    : 80.42%
    Last available       : 000000010000000000000009

  Catalog information:
    Retention Policy     : not enforced
    Previous Backup      : - (this is the oldest base backup)
    Next Backup          : - (this is the latest base backup)
```

> Simulation d'un incident : supprimer tout le contenu de la table.

```
TRUNCATE TABLE matable;
```

> Restaurer les données avant l'incident à l'aide de barman.

Arrêter l'instance PostgreSQL. Pour le TP, on peut renommer le PGDATA mais il n'est pas nécessaire de le supprimer vous-même.

Il faut savoir que `--remote-ssh-command` est nécessaire, sinon barman tentera de restaurer

un PGDATA sur son serveur et avec ses droits.

Pour éviter de devoir configurer la connexion SSH, nous pouvons autoriser l'utilisateur système `barman` à faire des modifications dans le répertoire `/var/lib/pgsql/14`. Par exemple :

```
# chmod 777 /var/lib/pgsql/
# chmod 777 /var/lib/pgsql/14
```

Lancer la commande de restauration en tant que **barman** :

```
barman$ barman recover \
--target-time "20211111 15:40:00" \
--target-action "promote" \
localhost 20211111T153507 /var/lib/pgsql/14/data

Starting local restore for server localhost using backup 20211111T153507
Destination directory: /var/lib/pgsql/14/data
Doing PITR. Recovery target time: '2021-11-11 15:40:00+01:00'
Copying the base backup.
Copying required WAL segments.
Generating recovery configuration
Identify dangerous settings in destination directory.
Recovery completed (start time: 2021-11-11 15:59:13.697531, elapsed time: 1 second)
Your PostgreSQL server has been successfully prepared for recovery!
```

Rétablir les droits sur le répertoire nouvellement créé par barman :

```
 # chown -R postgres: /var/lib/pgsql/14/data
```

Démarrer PostgreSQL.

Vérifier les logs et la présence de la table disparue.

```
$ cat /var/lib/pgsql/14/data/log/postgresql-Wed.log

[…]
2021-11-11 16:01:21.699 CET [28525] LOG:  redo done at 0/9D49D68
2021-11-11 16:01:21.699 CET [28525] LOG:  last completed transaction was
                                       at log time 2021-11-11 15:36:08.184735+01
2021-11-11 16:01:21.711 CET [28525] LOG:  restored log file
                                       "000000010000000000000009" from archive
2021-11-11 16:01:21.777 CET [28525] LOG:  selected new timeline ID: 2
2021-11-11 16:01:21.855 CET [28525] LOG:  archive recovery complete
2021-11-11 16:01:22.043 CET [28522] LOG:  database system is ready to
                                                        accept connections

SELECT count(*) FROM matable ;
```

```
  count
---------
 1000000
```

Avant de passer à la suite de la formation, pour stopper les commandes démarrées par `barman cron` :

```
barman$ barman receive-wal --stop localhost
```

Il est possible de vérifier la liste des serveurs sur lesquels appliquer cette modification à l'aide de la commande `barman list-server`.

Pour désactiver totalement barman :

```
$ mv /etc/barman.d/localhost.conf /etc/barman.d/localhost.conf.old
$ sudo -iu barman barman cron
```

1.10.3 UTILISATION DE PITRERY (OPTIONNEL)

> Installer pitrery.

Les développeurs de pitrery mettent à disposition les paquets RPM dans un dépôt spécifique (https://yum.dalibo.org/labs/).

Pour installer ce dépôt (ici sous Rocky Linux ; utiliser `yum` sous CenOS 7) :

```
# dnf install -y https://yum.dalibo.org/labs/dalibo-labs-4-1.noarch.rpm
# dnf repolist
```

Il est également possible de l'installer en compilant les sources.

La commande suivante devrait dès lors suffire :

```
# yum install pitrery
```

> Configurer pitrery pour la sauvegarde du serveur local.

Par défaut, le fichier de configuration créé est `/etc/pitrery/pitrery.conf`. Créons-en une copie vide.

```
# cp /etc/pitrery/pitrery.conf /etc/pitrery/pitrery.conf.bck
# echo > /etc/pitrery/pitrery.conf
```

Configurons-le ensuite de la manière suivante :

```
####################
# Backup management
```

```
####################
PGDATA="/var/lib/pgsql/14/data"
PGUSER="postgres"
BACKUP_DIR="/var/lib/pgsql/14/backups/pitr"
PURGE_KEEP_COUNT=1

####################
# WAL archiving
####################
ARCHIVE_DIR="$BACKUP_DIR/archived_wal"
```

Ce fichier indique où se trouve notre répertoire de données `PGDATA`, les informations de connexion (ici uniquement `PGUSER`) ainsi que la configuration relative au stockage des backups et des journaux de transactions archivés.

Créer les répertoires `BACKUP_DIR` et `ARCHIVE_DIR` avec l'utilisateur **postgres** :

```
$ mkdir -p /var/lib/pgsql/14/backups/pitr/archived_wal
```

Il convient ensuite de modifier la configuration du fichier `postgresql.conf` ainsi :

```
wal_level = replica
archive_mode = on
archive_command = '/usr/bin/archive_wal %p'
```

pitrery fournit le script `archive_wal` pour gérer la commande d'archivage. Par défaut, ce script utilisera le `pitrery.conf` que nous venons de configurer.

Il faut redémarrer le service PostgreSQL si les paramètres `wal_level` ou `archive_mode` ont été modifiés, sinon un simple rechargement de la configuration suffit.

Il est préférable de tester que la configuration est bonne. Cela se fait avec cette commande :

```
$ pitrery check

INFO: checking /etc/pitrery/pitrery.conf
INFO: the configuration file contains:
PGDATA="/var/lib/pgsql/14/data"
PGUSER="postgres"
BACKUP_DIR="/var/lib/pgsql/14/backups/pitr"
PURGE_KEEP_COUNT=1
ARCHIVE_DIR="$BACKUP_DIR/archived_wal"

INFO: ==> checking the configuration for inconsistencies
INFO: configuration seems correct
INFO: ==> checking backup configuration
INFO: backups are local, not checking SSH
```

```
INFO: target directory '/var/lib/pgsql/14/backups/pitr' exists
INFO: target directory '/var/lib/pgsql/14/backups/pitr' is writable
INFO: ==> checking WAL files archiving configuration
INFO: WAL archiving is local, not checking SSH
INFO: checking WAL archiving directory: /var/lib/pgsql/14/backups/pitr/archived_wal
INFO: target directory '/var/lib/pgsql/14/backups/pitr/archived_wal' exists
INFO: target directory '/var/lib/pgsql/14/backups/pitr/archived_wal' is writable
INFO: ==> checking access to PostgreSQL
INFO: psql command and connection options are: psql -X -U postgres
INFO: connection database is: postgres
INFO: environment variables (maybe overwritten by the configuration file):
INFO:   PGUSER=postgres
INFO:   PGDATA=/var/lib/pgsql/14/data
INFO: PostgreSQL version is: 14.1
INFO: connection role can run backup functions
INFO: current configuration:
INFO:   wal_level = replica
INFO:   archive_mode = on
INFO:   archive_command = '/usr/bin/archive_wal %p'
INFO: ==> checking access to PGDATA
INFO: PostgreSQL and the configuration reports the same PGDATA
INFO: permissions of PGDATA ok
INFO: owner of PGDATA is the current user
INFO: access to the contents of PGDATA ok
```

> Faire une sauvegarde.

```
$ pitrery backup

INFO: preparing directories in /var/lib/pgsql/14/backups/pitr/
INFO: listing tablespaces
INFO: starting the backup process
INFO: performing a non-exclusive backup
INFO: backing up PGDATA with tar
INFO: archiving /var/lib/pgsql/14/data
INFO: stopping the backup process
INFO: copying the backup history file
INFO: copying the tablespaces list
INFO: copying PG_VERSION
INFO: backup directory is /var/lib/pgsql/14/backups/pitr/2021.11.26_12.10.56
INFO: done
```

> Lister les sauvegardes.

```
$ pitrery list
```

```
List of local backups
/var/lib/pgsql/14/backups/pitr/2021.11.26_12.10.56  3.2M    2021-11-26 12:10:56 CET
```

> Faire une restauration.

Le service doit être arrêté et le répertoire `PGDATA` supprimé. Un message `FATAL` prévient lorsqu'un répertoire cible n'est pas vide : `/var/lib/pgsql/14/data is not empty. Contents won't be overwritten`.

```
 # systemctl stop postgresql-14

$ rm -rf /var/lib/pgsql/14/data

$ pitrery restore

INFO: searching backup directory
INFO: retrieving the PostgreSQL version of the backup
INFO: PostgreSQL version: 14
INFO: searching for tablespaces information
INFO:
INFO: backup:
INFO:   directory: /var/lib/pgsql/14/backups/pitr/2021.11.26_12.10.56
INFO:   storage: tar
INFO:   encryption: no
INFO:
INFO: destinations directories:
INFO:   PGDATA -> /var/lib/pgsql/14/data
INFO:
INFO: recovery configuration:
INFO:   target owner of the restored files: postgres
INFO:   restore_mode = 'recovery'
INFO:   restore_command = 'restore_wal -C /etc/pitrery/pitrery.conf %f %p'
INFO:
INFO: creating /var/lib/pgsql/14/data with permission 0700
INFO: extracting PGDATA to /var/lib/pgsql/14/data
INFO: extraction of PGDATA successful
INFO: preparing pg_wal directory
INFO: Create the /var/lib/pgsql/14/data/recovery.signal file (recovery mode)
INFO: preparing postgresql.conf file
INFO: done
INFO:
INFO: please check directories and recovery configuration before starting the
INFO: cluster and do not forget to update the configuration of pitrery if
INFO: needed:
INFO:   /var/lib/pgsql/14/data/postgresql.conf
INFO:
```

NOTES

NOTES

NOTES

NOTES

NOS AUTRES PUBLICATIONS

FORMATIONS

- **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

LIVRES BLANCS

- **Migrer d'Oracle à PostgreSQL**
- **Industrialiser PostgreSQL**
- **Bonnes pratiques de modélisation avec PostgreSQL**
- **Bonnes pratiques de développement avec PostgreSQL**

TÉLÉCHARGEMENT GRATUIT

Les versions électroniques de nos publications sont disponibles gratuitement sous licence open-source ou sous licence Creative Commons. Contactez-nous à l'adresse contact@dalibo.com pour plus d'information.

DALIBO, L'EXPERTISE POSTGRESQL

Depuis 2005, DALIBO met à la disposition de ses clients son savoir-faire dans le domaine des bases de données et propose des services de conseil, de formation et de support aux entreprises et aux institutionnels.

En parallèle de son activité commerciale, DALIBO contribue aux développements de la communauté PostgreSQL et participe activement à l'animation de la communauté francophone de PostgreSQL. La société est également à l'origine de nombreux outils libres de supervision, de migration, de sauvegarde et d'optimisation.

Le succès de PostgreSQL démontre que la transparence, l'ouverture et l'auto-gestion sont à la fois une source d'innovation et un gage de pérennité. DALIBO a intégré ces principes dans son ADN en optant pour le statut de SCOP : la société est contrôlée à 100 % par ses salariés, les décisions sont prises collectivement et les bénéfices sont partagés à parts égales.