Module I4

Outils de sauvegarde physique

24.12

Table des matières

Sur ce document

Formation	Module I4
Titre	Outils de sauvegarde physique
Révision	24.12
PDF	https://dali.bo/i4_pdf
EPUB	https://dali.bo/i4_epub
HTML	https://dali.bo/i4_html
Slides	https://dali.bo/i4_slides
TP	https://dali.bo/i4_tp
TP (solutions)	https://dali.bo/i4_solutions

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

Chers lectrices & lecteurs,

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

Au-delà du contenu technique en lui-même, notre intention est de transmettre les valeurs qui animent et unissent les développeurs de PostgreSQL depuis toujours : partage, ouverture, transparence, créativité, dynamisme… Le but premier de nos formations est de vous aider à mieux exploiter toute la puissance de PostgreSQL mais nous espérons également qu'elles vous inciteront à devenir un membre actif de la communauté en partageant à votre tour le savoir-faire que vous aurez acquis avec nous.

Nous mettons un point d'honneur à maintenir nos manuels à jour, avec des informations précises et des exemples détaillés. Toutefois malgré nos efforts et nos multiples relectures, il est probable que ce document contienne des oublis, des coquilles, des imprécisions ou des erreurs. Si vous constatez un souci, n'hésitez pas à le signaler via l'adresse formation@dalibo.com[1] !

À propos de DALIBO

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

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

[1] mailto:formation@dalibo.com

Remerciements

Ce manuel de formation est une aventure collective qui se transmet au sein de notre société depuis des années. Nous remercions chaleureusement ici toutes les personnes qui ont contribué directement ou indirectement à cet ouvrage, notamment :

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

Forme de ce manuel

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

Licence Creative Commons CC-BY-NC-SA

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

Marques déposées

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

Versions de PostgreSQL couvertes

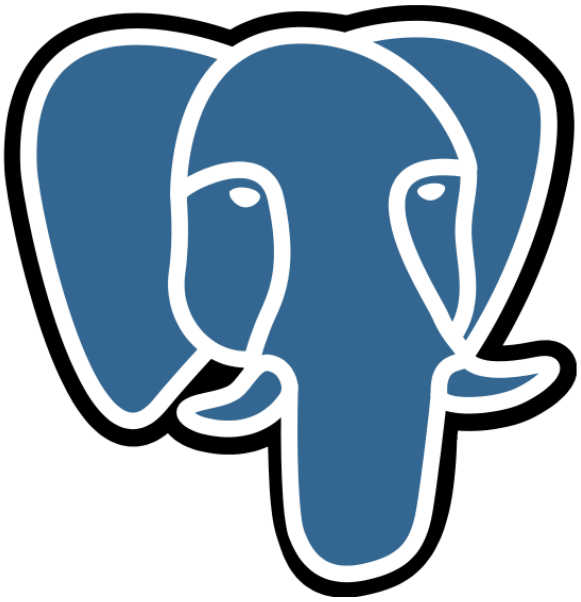
Ce document ne couvre que les versions supportées de PostgreSQL au moment de sa rédaction, soit les versions 13 à 17.

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

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

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

1/ 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
- 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 Préalable : définir les besoins

- 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

1.2.1 pg_basebackup - Présentation

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

pg_basebackup[1] 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.

1.2.2 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.

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`.

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

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.

1.2.3 pg_basebackup - Avantages

- Transfert des WAL pendant la sauvegarde
- Slot de réplication automatique (temporaire voire permanent)
- Limitation du débit
- Relocalisation des tablespaces
- Fichier manifeste
- Vérification des checksums
- Sauvegarde possible à partir d'un secondaire
- Compression côté serveur ou client (v15+)
- Emplacement de la sauvegarde (client/server/blackhole) (v15+)
- Suivi : `pg_stat_progress_basebackup`

pg_basebackup s'est beaucoup amélioré au fil des versions et son comportement a parfois changé. Regardez bien la documentation[2] 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 entre-temps (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 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.

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.

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

À l'inverse, `-X none` peut être utile si la récupération des journaux est réalisée par ailleurs (généralement par `archive_command` ou `archive_library`). 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 initialisé mais 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 :

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).

Emplacement de la sauvegarde

À partir de la version 15, l'option `--target` permet de spécifier où la sauvegarde doit être réalisée :

- sur le serveur où la commande est lancée (`client`) ;
- sur le serveur de base de données (`server`) ;
- dans le vide (`blackhole`).

Des destinations peuvent être ajoutées par des extensions, `basebackup_to_shell`[3] est fournie à titre d'exemple et permet d'exécuter une commande à l'issue d'une sauvegarde.

[3] https://docs.postgresql.fr/current/basebackup-to-shell.html

Lorsque la destination `server` est choisie, plusieurs restrictions s'appliquent à la sauvegarde :

- le format doit être `tar` ;
- l'utilisateur employé pour la réaliser doit être membre du rôle `pg_write_server_files` ;
- la méthode de récupération des WAL doit être `fetch` ou `none`.

Compression de la sauvegarde :

À partir de la version 15, il est possible de demander la compression de la sauvegarde avec un grand niveau de personnalisation :

- algorithme de compression parmi `gzip`, `lz4` et `zstd` ;
- rapidité de la compression hors parallélisme (`lz4`) ;
- niveau de compression (`zstd`) ;
- parallélisation de la compression (`zstd`) ;
- localisation de la compression (serveur ou client).

Cela permet de gérer différents scénarios et d'éviter certains goulets d'étranglement lors d'une sauvegarde.

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 15) :

```
$ pg_basebackup --help
```

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

```
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, --target=CIBLE[:DETAIL]    cible de sauvegarde (si autre que client)
  -T, --tablespace-mapping=ANCIENREP=NOUVEAUREP
                                 déplace le répertoire ANCIENREP en NOUVEAUREP
      --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=[{client|server}-]METHODE[:DETAIL]
                                 compresse sur le client ou le serveur comme indiqué
  -Z, --compress=none            ne compresse pas la sortie tar

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/>
```

1.2.4 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
  - incrémentale (si PostgreSQL 17 avec `pg_combinebackup`)
- 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` ou `archive_library` 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`).

Il ne sait faire des sauvegardes incrémentales qu'à partir de PostgreSQL 17. Les archives créées sont à restaurer avec le nouvel outil `pg_combinebackup` , dont le maniement est encore assez fastidieux.

Restauration :

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

La copie est directement utilisable, éventuellement après déplacement et/ou décompression des `.tar.gz` . Mais, généralement, on ajoutera un fichier `recovery.signal` , et on définira la `restore_command` pour récupérer les archives. Dans l'idéal, `restore_command` sera déjà prête dans le `postgresql.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 (`postgresql.auto.conf` et fichier vide `standby.signal`). avec les paramètres pour une réplication en *streaming*.

1.3 PGBACKREST

1.3.1 pgBackRest - Présentation générale

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

1.3.2 pgBackRest - Fonctionnalités

- Gère la sauvegarde et la restauration
  - *pull* ou *push*, multidépôts
  - mono- ou multiserveur
- Indépendant des commandes système
  - protocole dédié
- Sauvegardes complètes, différentielles ou incrémentales
- Multithread, sauvegarde depuis un secondaire, archivage asynchrone…
- Projet mature

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

[5] https://pgbackrest.org/

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 multithread ;
- 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, ou avoir plusieurs niveaux d'archives) ;
- le support intégré de dépôts S3 ou Azure ;
- le support d'un accès TLS géré par pgBackRest en alternative à SSH ;
- 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 ;
- la reprise d'une sauvegarde échouée.

pgBackRest n'utilise pas `pg_receivewal` pour garantir la sauvegarde du dernier journal (non terminé) avant un sinistre. Les auteurs considèrent que dans ce cas un secondaire synchrone est plus adapté et plus fiable.

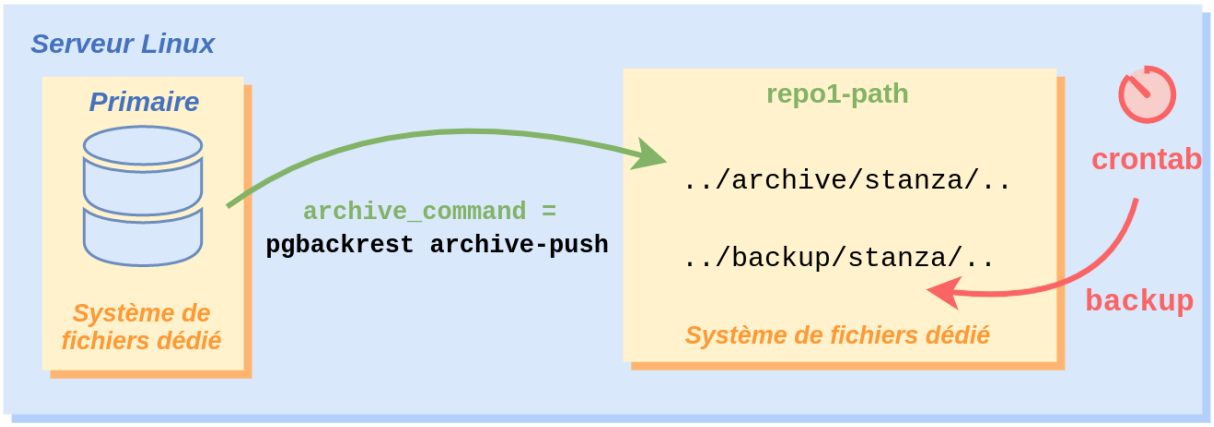
Le projet est très actif et considéré comme fiable, et les fonctionnalités proposées sont intéressantes.

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

1.3.3 pgBackRest - Sauvegardes

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

[6] https://github.com/pgstef/check_pgbackrest/



pgBackRest gère uniquement des sauvegardes physiques.

La sauvegarde s'effectue :

- soit en local (*push*, directement sur le serveur hébergeant l'instance à sauvegarder) pour un stockage local des sauvegardes, ou un stockage accessible par un montage NFS ou vers un dépôt S3, Azure… ;
- soit depuis un serveur distant (*pull*), déléguant ainsi l'ordonnancement et le stockage des données à celui-ci.

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

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

L'archivage des journaux se fait bien sûr en permanence et utilise le classique `archive_command`.

1.3.4 pgBackRest - Restauration

- Depuis le serveur de BDD avec un dépôt local 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()`.

1.3.5 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/

1.3.6 pgBackRest - Utilisation

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

Commands:
    annotate         Add or modify backup annotation.
    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.
    repo-get         Get a file from a repository.
    repo-ls          List files in a repository.
    restore          Restore a database cluster.
    server           pgBackRest server.
    server-ping      Ping pgBackRest server.
    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.
    verify           Verify contents of the repository.
    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.

1.3.7 pgBackRest - Configuration

- `/etc/pgbackrest.conf`
- Configuration générale dans la section `[global]`
- Chaque instance à sauvegarder doit avoir sa propre section, appelée `stanza`
- possibilité d'éclater la configuration dans plusieurs fichiers : `config-include-path`

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

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

Il peut y avoir plusieurs stanzas déclarées dans le fichier, notamment s'il est situé sur le serveur où sont stockées les sauvegardes de plusieurs instances.

Pour des questions de lisibilité, il est possible de créer un fichier de configuration par instance à sauvegarder. Le nom du fichier doit se terminer par `.conf` pour être pris en compte. Les fichiers doivent être regroupés dans un répertoire référencé par le paramètre `config-include-path`.

1.3.8 pgBackRest - Configuration PostgreSQL

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

```
archive_mode = on
wal_level = replica
archive_command = 'pgbackrest --stanza=erp_prod archive-push %p'
archive_timeout = '? min'   # à définir
```

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.

Comme pgBackRest n'archive que des journaux complets, il vaut mieux penser à mettre un `archive_timeout` adapté au RPO accepté. (S'il est nul, les auteurs recommandent plutôt un secondaire synchrone[9]).

1.3.9 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.

[9] https://github.com/pgbackrest/pgbackrest/issues/2233#issuecomment-1831406999

1.3.10 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 le 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.

1.3.11 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 ;
- …

1.3.12 pgBackRest - Configuration TLS

- Alternative au SSH
- `{repo1|pg1}-host-type = tls`
- paramètres `tls-server-{address|auth|cert|key|ca}`
- paramètres `repo1-host-{cert|key|ca}`
- paramètres `pg1-host-{cert|key|ca}`
- `pgbackrest server`

Il existe une alternative à l'utilisation de SSH qui consiste à configurer un serveur TLS en valorisant le paramètre `repo1-host-type` et `pg1-host-type` à `tls` (défaut : `ssh`). La configuration du serveur se fait ensuite avec les paramètres :

- `tls-server-address` : adresse IP sur laquelle le serveur écoute pour servir des requêtes clients ;
- `tls-server-auth` : la liste des clients autorisés à se connecter sous la forme `<client-cn>=<stanza>` ;
- `tls-server-ca-file` : certificat de l'autorité ;
- `tls-server-cert-file` : certificat du serveur ;
- `tls-server-key-file` : clé du serveur.

Il faut ensuite configurer l'accès au dépôt de sauvegarde :

- `repo1-host-type=tls` : la connexion au dépôt utilise TLS ;
- `repo1-host-cert-file` : certificat pour se connecter au dépôt ;
- `repo1-host-key-file` : clé pour se connecter au dépôt ;
- `repo1-host-ca-file` : certificat de l'autorité.

Exemple de configuration :

```
[global]
repo1-host=backrest-srv
repo1-host-user=backrest
repo1-host-type=tls
repo1-host-cert-file=/etc/certs/srv1-cert.pem
repo1-host-key-file=/etc/certs/srv1-key.pem
repo1-host-ca-file=/etc/certs/CA-cert.pem

tls-server-address=*
tls-server-cert-file=/etc/certs/srv1-cert.pem
tls-server-key-file=/etc/certs/srv1-key.pem
tls-server-ca-file=/etc/certs/CA-cert.pem
tls-server-auth=backrest-srv=main

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

Sur le serveur de sauvegarde, la configuration est similaire :

- `pg1-host-type=tls` : la connexion au serveur PostgreSQL utilise TLS ;
- `pg1-host-cert-file` : certificat pour se connecter au serveur de bases de données ;
- `pg1-host-key-file` : certificat pour se connecter au serveur de bases de données ;
- `pg1-host-ca-file` : certificat de l'autorité.

Exemple de configuration du serveur de sauvegarde :

```
[global]
repo1-path=/var/lib/pgbackrest
repo1-retention-full=2

tls-server-address=*
tls-server-cert-file=/etc/certs/backrest-srv-cert.pem
tls-server-key-file=/etc/certs/backrest-srv-key.pem
tls-server-ca-file=/etc/certs/CA-cert.pem
tls-server-auth=srv1=main

[main]
pg1-host=srv1
pg1-port=5432
pg1-path=/var/lib/pgsql/14/data

pg1-host-type=tls
pg1-host-cert-file=/etc/certs/backrest-srv-cert.pem
pg1-host-key-file=/etc/certs/backrest-srv-key.pem
pg1-host-ca-file=/etc/certs/CA-cert.pem
```

Le serveur TLS doit ensuite être démarré avec la commande `pgbackrest server`. Un service est prévu à cet effet et installé automatiquement sur les distributions de type RedHat et Debian.

Un ping vers le serveur TLS peut être testé avec la commande `pgbackrest server-ping <hote>`. Suivant les distributions, il peut être nécessaire d'ouvrir le port 8432 (valeur par défaut de `tls-server-port`).

```
[postgres@backrest log]$ pgbackrest server-ping srv1
INFO: server-ping command begin 2.41: [srv1] --exec-id=7467-76e4b8cf
 --log-level-console=info --tls-server-address=*
INFO: server-ping command end: completed successfully (47ms)
```

Génération des clés et certificats auto-signés :

```
# Générer une clé privée et un certificat pour l'autorité de certification
openssl req -new -x509 \
                 -days 365 \
         -nodes \
         -out CA-cert.pem \
         -keyout CA-key.pem \
         -subj "/CN=root-ca"

# Générer une clé privée et demande de certificat (CSR)
openssl req -new -nodes \
                 -out backrest-srv-csr.pem \
         -keyout backrest-srv-key.pem \
         -subj "/CN=backrest-srv"
openssl req -new -nodes \
                 -out srv1-csr.pem \
         -keyout srv1-key.pem \
         -subj "/CN=srv1"

# Générer le certificat signé
openssl x509 -req -in backrest-srv-csr.pem \
                  -days 365 \
          -CA CA-cert.pem \
          -CAkey CA-key.pem \
          -CAcreateserial \
          -out backrest-srv-crt.pem
openssl x509 -req -in srv1.csr
                  -days 365 \
          -CA CA-cert.pem \
          -CAkey CA-key.pem \
          -CAcreateserial \
          -out srv1-crt.pem
```

1.3.13 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.

1.3.14 pgBackRest - Exemple configuration par instance

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

```
[erp_prod]
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 `postgresql.auto.conf` qui sera généré automatiquement à la restauration d'une sauvegarde.

1.3.15 pgBackRest - Initialiser le répertoire de stockage des sauvegardes

- Pour initialiser le répertoire de stockage des sauvegardes

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

- Vérifier la configuration de l'archivage

```
$ sudo -u postgres pgbackrest --stanza=erp_prod 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.

1.3.16 pgBackRest - Effectuer une sauvegarde

- Pour déclencher une nouvelle sauvegarde complète

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

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

La sauvegarde accepte de nombreux paramètres dont :

- `--archive-copy` : archive les WAL dans la sauvegarde en plus de les mettre dans le dépôt de WAL ;
- `--backup-standby` : déclenche la sauvegarde sur un serveur secondaire ;
- `--no-online` : fait une sauvegarde à froid ;
- `--resume` : reprend une sauvegarde précédemment échouée en conservant les fichiers qui n'ont pas changés ;
- `--start-fast` : exécuter le checkpoint immédiatement.

Exemple de sortie d'une sauvegarde complète :

```
$ sudo -u postgres pgbackrest --stanza=erp_prod --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=erp_prod --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=erp_prod --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.

Il est possible d'ajouter des annotations aux sauvegardes comme ceci :

```
$ sudo -u postgres pgbackrest
  --stanza=erp_prod
  --type=full
  --annotation=desc="Premier backup"
  backup
```

L'annotation peut être observé en affichant les informations du *backup set*.

1.3.17 pgBackRest - Lister les sauvegardes

- Lister les sauvegardes présentes et leur taille

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

- ou une sauvegarde spécifique (*backup set*)

```
$ sudo -u postgres pgbackrest --stanza=erp_prod --set 20221026-071751F
 ↪  info
```

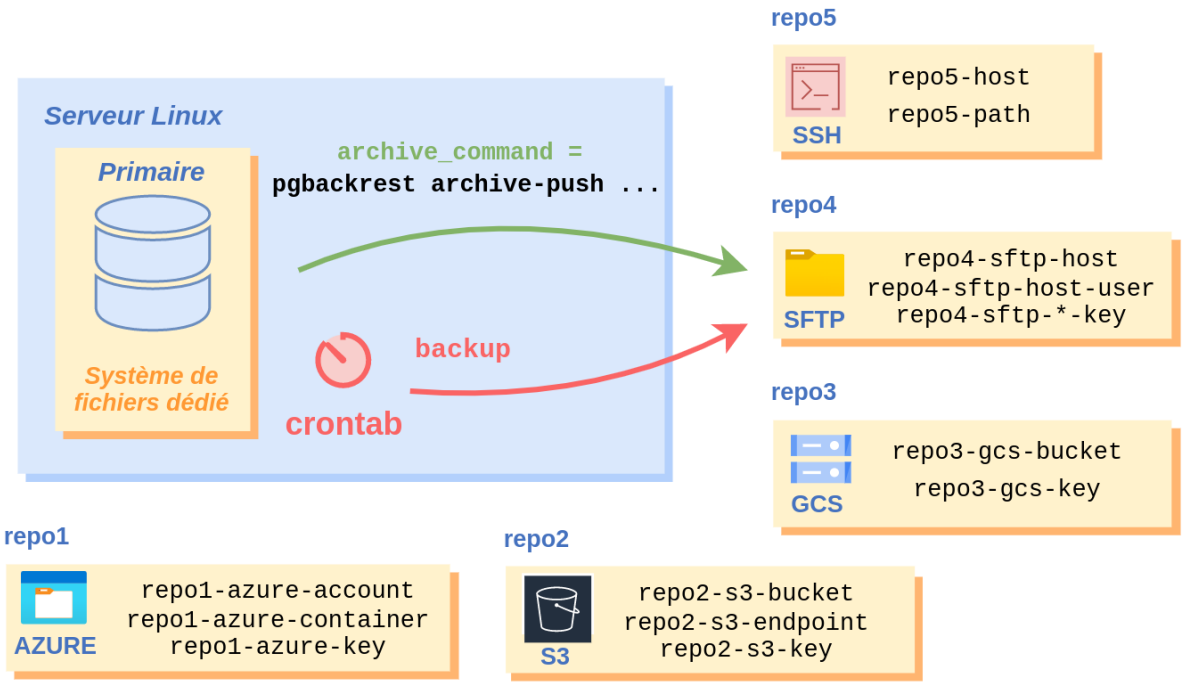
Exemple de sortie des commandes :

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

    db (current)
        wal archive min/max (14): 000000030000000000000019/00000003000000000000001B

        full backup: 20221026-071751F
            timestamp start/stop: 2022-10-26 07:17:51 / 2022-10-26 07:17:57
            wal start/stop: 00000003000000000000001B / 00000003000000000000001B
            database size: 25.2MB, database backup size: 25.2MB
            repo1: backup set size: 3.2MB, backup size: 3.2MB

$ sudo -u postgres pgbackrest --stanza=erp_prod --set 20221026-071751F info
stanza: erp_prod
    status: ok
    cipher: none

    db (current)
        wal archive min/max (14): 000000030000000000000019/00000003000000000000001B

        full backup: 20221026-071751F
            timestamp start/stop: 2022-10-26 07:17:51 / 2022-10-26 07:17:57
            wal start/stop: 00000003000000000000001B / 00000003000000000000001B
            lsn start/stop: 0/1B000028 / 0/1B000100
            database size: 25.2MB, database backup size: 25.2MB
            repo1: backup set size: 3.2MB, backup size: 3.2MB
            database list: postgres (13748)
```

```
annotation(s)
    desc: Premier backup
```

1.3.18 pgBackRest - Dépôts

- Plusieurs dépôts simultanés possibles
  - sauvegarde par dépôt selon rétention
  - archivage sur tous les dépôts (asynchrone conseillé!)
  - `--repo1-option=…` , appel avec `--repo=1`
- POSIX (NFS, ssh), CIFS, SFTP, cloud (S3, Azure, GFS)



pgBackRest permet de maintenir plusieurs dépôts de sauvegarde simultanément[10].

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.

[10] https://pgbackrest.org/configuration.html#section-repository

Une sauvegarde se fait vers un seul dépôt donné en le désignant explicitement. Cependant, l'archivage des journaux est simultané sur tous les dépôts à la fois. 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[11]), 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.

Pour les détails, voir la conférence de Stefan Fercot à la PGSession 16 de 2021[12] (slides[13].

[11] https://docs.postgresql.fr/current/creating-cluster.html#CREATING-CLUSTER-FILESYSTEM
[12] https://dali.bo/pgsession14_conf_pgbackrest
[13] https://pgsessions.com/assets/archives/pgs14_depots_sauvegarde_multiples_avec_pgBackRest.pdf%5D

1.3.19 pgBackRest - bundling et sauvegarde incrémentale en mode block

- Regrouper les petits fichiers dans des bundles

```
repo1-bundle=y
```

- Sauvegarde incrémentale en mode block (requiert le bundling)

```
repo1-bundle=y
repo1-block=y
```

« Bundling » des petits fichiers

Si une instances contient de nombreux petits fichiers (base aux nombreuses toutes petites tables, `pg_commit_ts` rempli à cause de `track_commit_timestamp` à `on`, très nombreuses petites partitions, chacune avec des fichiers annexes…), il est possible de les regrouper par paquets.

```
repo1-bundle=y
# défauts
repo1-bundle-limit=2MiB
repo-bundle-size=20MiB
```

Les bundles ne sont pas conservés en cas de backup interrompu puis redémarré. Les fichiers doivent être re-sauvegardés lors de la relance. Bundles et hard-links ne peuvent pas être utilisés ensemble.

Cette fonctionnalité est particulièrement utile avec un stockage comme S3 où le coût de création de fichier est prohibitif.

Sauvegarde incrémentale en mode bloc (2.46)

La sauvegarde incrémentale par bloc permet plus de granularité en divisant les fichiers en blocs qui peuvent être sauvegardés indépendamment. C'est particulièrement intéressant pour des fichiers avec peu de modifications, car pgBackRest ne sauvegardera que quelques blocs au lieu du fichier complet (les tables et index sont segmentés en fichiers de 1 Go). Cela permet donc d'économiser de l'espace dans le dépôt de sauvegarde et accélère les restaurations par delta.

La sauvegarde incrémentale par bloc doit être activée sur tous les types de sauvegardes : full, incrémentielle ou différentielle. Cela aura pour impact de rendre la sauvegarde full un peu plus grosse du fait de la création de fichier de cartographie des blocs. En revanche, les sauvegardes différentielles et incrémentielles suivantes pourront utiliser cette cartographie pour économiser de l'espace.

La taille du bloc pour un fichier donné est définie en fonction de l'âge et de la taille du fichier. Généralement, les fichiers les plus gros et/ou les plus anciens auront des tailles de bloc supérieures. Si un fichier est assez vieux, aucune cartographie ne sera crée.

Cette fonctionnalité nécessite le *bundling* et s'active ainsi :

```
repo1-block=y
repo1-bundle=y
```

1.3.20 pgBackRest - Restauration

- Effectuer une restauration

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

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

Exemple de sortie de la commande :

```
$ sudo -u postgres pgbackrest --stanza=erp_prod 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=erp_prod
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
```

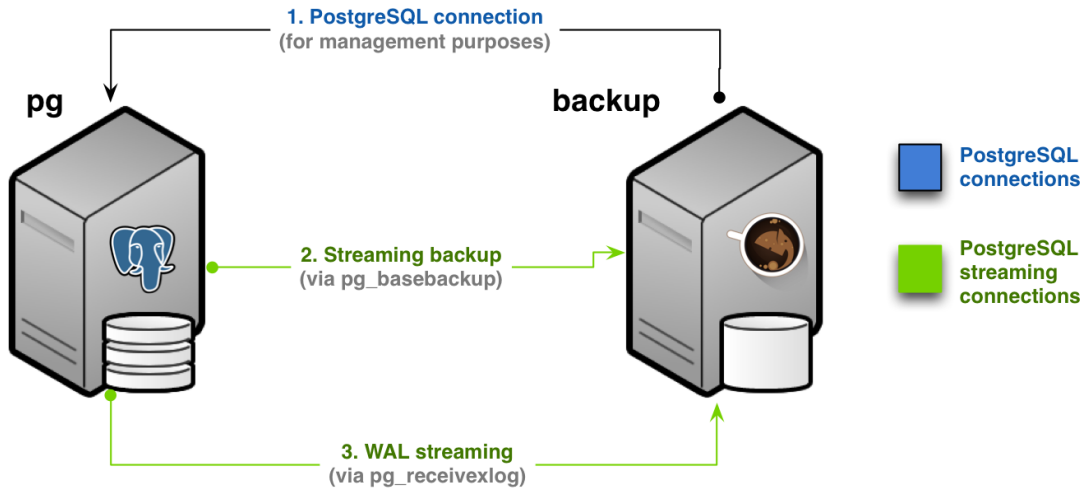
1.4 BARMAN

1.4.1 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.

1.4.2 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. Il faudra quand même configurer le SSH pour rendre possible la restauration depuis un serveur dédié ou faciliter la restauration en local.

En effet, les outils `pg_basebackup` et `pg_receivewal` sont utilisés pour ces opérations et se basent donc uniquement sur le protocole de réplication. Cela a pour avantage que les améliorations faites aux outils dans le cadre des mises à jour majeures de PostgreSQL sont disponible directement dans Barman.

Par exemple :

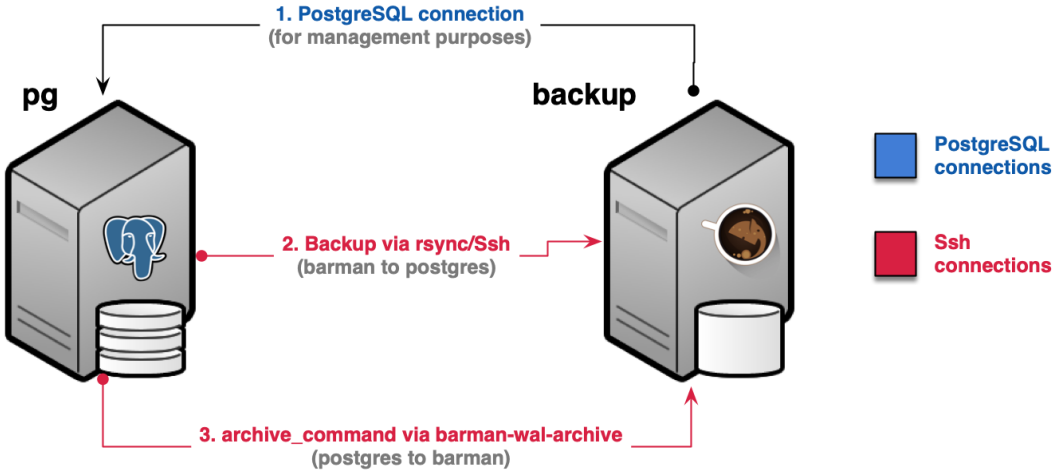
- la possibilité d'utiliser `pg_stat_progress_basebackup` pour la supervision ;
- les fichiers manifestes de sauvegarde et la vérification des sauvegardes ;
- la compression des sauvegardes.

Afin de garantir que l'instance sauvegardée conserve bien les WAL nécessaires, Barman permet de créer automatiquement un slot de réplication. Il se chargera également de démarrer `pg_receivewal` grâce à sa tâche de maintenance programmée en crontab.

L'archivage peut être configuré à la place ou en plus du streaming des WAL.

Ce mode de sauvegarde permet de sauvegarder un serveur PostgreSQL installé sous Windows.

1.4.3 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.

Cette méthode ne permet pas de compresser les sauvegardes mais permet de faire de la déduplication avec des *hard links* et de bénéficier de la parallélisation.

1.4.4 Barman - Sauvegardes

- Type de sauvegarde : **physique/PITR** (à chaud)
- Type de stockage : **local** ou **pull**
- Planification : **crontab**
- Méthodes :
  - **pg_backup_start() / rsync / pg_backup_stop()**
  - **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_backup_start()`, copie des fichiers, `pg_backup_stop()`.

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.

1.4.5 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.
- Compression des données via `pg_basebackup`

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.

1.4.6 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.

1.4.7 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()`.

1.4.8 Barman - Installation

- Accéder au dépôt communautaire PGDG
- Installer les paquets `barman` et `barman-cli`

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…)[14] ou rpm (Red Hat, Rocky Linux, CentOS, Fedora…)[15].

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.

[14] https://apt.postgresql.org/pub/repos/apt/
[15] https://yum.postgresql.org/

1.4.9 Barman - Utilisation

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

          {archive-wal,backup,check,check-backup,check-wal-archive,cron,

 ↪  delete,diagnose,generate-manifest,get-wal,help,keep,list-backup,
          list-backups,list-files,list-server,list-servers,put-wal,

 ↪  rebuild-xlogdb,receive-wal,recover,replication-status,show-backup,

 ↪  show-backups,show-server,show-servers,status,switch-wal,switch-xlog,
          sync-backup,sync-info,sync-wals,verify,verify-backup}

[...]
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.

1.4.10 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.

1.4.11 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.

1.4.12 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.

1.4.13 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 réper-

toire *incomming* adapté. Cela réduit donc le risque de corruption, perte de données ou simplement d'erreur de répertoire.

1.4.14 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`.

1.4.15 Barman - Configuration sauvegardes

- Configuration globale des options de sauvegarde

```
compression = gzip
backup_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`) ;
- **backup_compression** : méthode utilisée par Barman pour compresser la sauvegarde, les options disponibles dépendent de la version de PostgreSQL utilisée (la version 15 apporte beaucoup de nouveautés à ce niveau) ;

- **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.

1.4.16 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`).

1.4.17 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.

1.4.18 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>`.

1.4.19 Barman - Configuration d'un dépôt synchronisé

- Copie à l'identique du dépôt d'origine
- Sur le dépôt à synchroniser :
  - `primary_ssh_command`
- Commandes :
  - `barman sync-info --primary <instance> <ID-sauvegarde>`
  - `barman sync-backup <instance> <ID-sauvegarde>`
  - `barman sync-wal <instance>`

Barman permet de créer une copie d'un dépôt barman pour répondre à des besoins de redondance géographique. Il suffit pour cela de valoriser le paramètre `primary_ssh_command` pour que le serveur barman client se connecte au serveur principal et duplique les sauvegardes et WAL.

La commande `barman sync-info --primary <instance> <ID-sauvegarde>` permet d'afficher les informations de synchronisation. Le processus de copie est lancé automatiquement par la tâche de maintenance automatisée. Il est aussi possible de lancer la synchronisation manuellement pour une sauvegarde en particulier avec `barman sync-backup <instance> <ID-sauvegarde>` ou pour les WAL avec `barman sync-wal <instance>`.

1.4.20 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.

1.4.21 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.

1.4.22 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
create_slot = auto
slot_name = barman
```

- `barman replication-status pgsrv`

La commande `barman replication-status` permet d'afficher l'état de la réplication :

```
$ barman replication-status pgsrv
Status of streaming clients for server 'pgsrv':
  Current LSN on master: 0/140001B0
  Number of streaming clients: 1

  1. Async WAL streamer
     Application name: barman_receive_wal
     Sync stage      : 3/3 Remote write
     Communication   : Unix domain socket
     User name       : barman
     Current state   : streaming (async)
     Replication slot: barman
     WAL sender PID  : 29439
     Started at      : 2022-10-17 14:54:02.122742+00:00
     Sent LSN   : 0/140001B0 (diff: 0 B)
     Write LSN  : 0/140001B0 (diff: 0 B)
     Flush LSN  : 0/14000000 (diff: -432 B)
```

1.4.23 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}
$ sudo -u barman barman check {<instance> | all} --nagios
```

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.

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
  superuser or standard user with backup privileges: 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)
  backup minimum size: OK (33.6 MiB)
  wal maximum age: OK (no last_wal_maximum_age provided)
  wal size: OK (0 B)
  compression settings: OK
  failed backups: OK (there are 0 failed backups)
  minimum redundancy requirements: OK (have 2 backups, expected at least 0)
  pg_basebackup: OK
  pg_basebackup compatible: OK
  pg_basebackup supports tablespaces mapping: OK
  systemid coherence: OK
  pg_receivexlog: OK
  pg_receivexlog compatible: OK
  receive-wal running: OK
  archiver errors: OK
```

1.4.24 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)
```

1.4.25 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.

1.4.26 Barman - Nouvelle sauvegarde

- Pour déclencher une nouvelle sauvegarde

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

- 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.

L'option `--wait` permet d'attendre que les WAL soient archivés avant de rendre la main.

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
```

1.4.27 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
```

1.4.28 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)
```

1.4.29 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.

1.4.30 Barman - Conserver une sauvegarde

- Pour conserver une sauvegarde

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

- Pour relâcher une sauvegarde

```
$ sudo -u barman barman keep --release <instance> <ID-sauvegarde>
```

Il est possible de marquer une sauvegarde pour qu'elle soit conservée par barman quelle que soit la rétention configurée avec la commande `barman keep <instance> <ID-sauvegarde>`.

La sauvegarde peut être relâchée en ajoutant le paramètre `--release`.

1.4.31 Barman - Tâches de maintenance

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

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

- À planifier ! (vérifier `/etc/cron.d/barman`)

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.

1.4.32 Barman - Restauration

- Copie/transfert de la sauvegarde
- Copie/transfert des journaux de transactions
- Génère le paramétrage pour la restauration
- 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.

1.4.33 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.

1.4.34 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 AUTRES OUTILS DE L'ÉCOSYSTÈME

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

Du fait du dynamisme du projet, l'écosystème des outils autour de PostgreSQL est très changeant. À côté des outils évoqués 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[16]), rendus inutiles par l'évolution de PostgreSQL ces dernières années (comme walmgr, de la suite Skytools[17], ou OmniPITR[18]) ou simplement peu actifs et peu rencontrés en production (par exemple pg_rman[19], développé par NTT).

Pitrery[20], de Nicolas Thauvin, issu du labo R&D de Dalibo, est encore supporté jusque 2026 et jusque PostgreSQL 15 inclus, mais plus au-delà. Il visait la simplicité d'utilisation pour des bases de taille petite ou moyenne.

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

1.5.1 WAL-G - présentation

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

WAL-G[21] est une réécriture d'un ancien outil assez populaire, WAL-E, par Citus[22] 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 multiprocesseur par différents algorithmes[23] et

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

l'optimisation du temps de restauration. Il supporte aussi MySQL/MariaDB et SQL Server (et d'autres dans le futur).

1.6 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, 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 !

1.7 QUIZ

https://dali.bo/i4_quiz

1.8 TRAVAUX PRATIQUES

La version en ligne des solutions de ces TP est disponible sur https://dali.bo/i4_solutions.

1.8.1 Utilisation de pgBackRest (Optionnel)

But : Sauvegarder et restaurer avec pgBackRest

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/1/backups` ; - le nom de la stanza sera **instance_dev** ; - prévoir de 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 (`pg_switch_wal`) pour 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.

1.8.2 Utilisation de barman (Optionnel)

But : Sauvegarder et restaurer avec barman

1.8.3 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 pour garantir 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 TRAVAUX PRATIQUES (SOLUTIONS)

1.9.1 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`. Le nom de la stanza sera **instance_dev**. 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

[instance_dev]
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=instance_dev 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=instance_dev --log-level-console=info stanza-create
```

Vérifier la configuration de pgBackRest et de l'archivage :

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

Vérifier que l'archivage fonctionne :

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

```
SELECT * FROM pg_stat_archiver;
```

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

```
$ pgbackrest --stanza=instance_dev --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=instance_dev --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=instance_dev --type=full
P00   INFO: expire command end: completed successfully (8ms)
```

Lister les sauvegardes :

```
$ pgbackrest --stanza=instance_dev info
stanza: instance_dev
    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=instance_dev --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=instance_dev --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.
```

1.9.2 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/000000010000000000000A.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 :

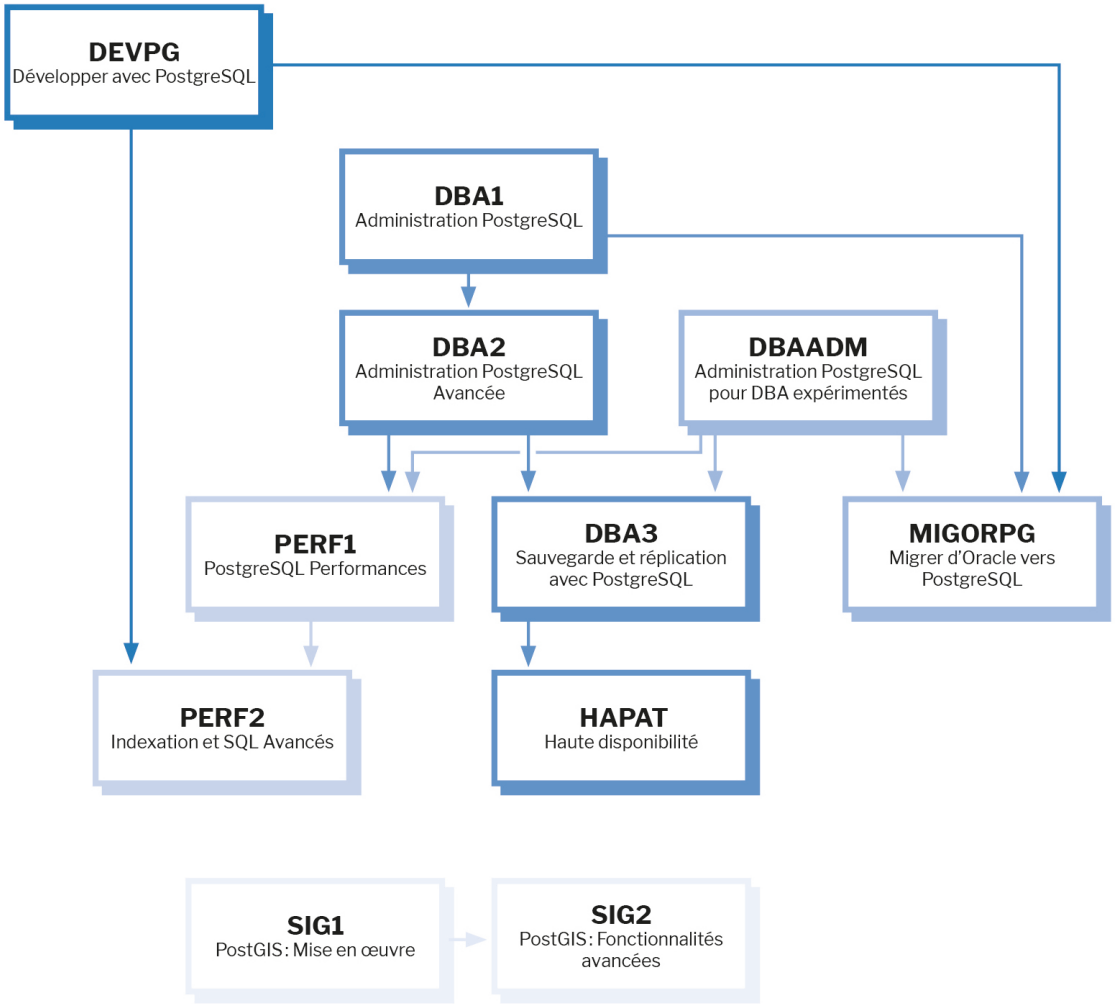
```
$ mv /etc/barman.d/localhost.conf /etc/barman.d/localhost.conf.old
$ sudo -iu barman barman cron
```

Les formations Dalibo

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

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

Cursus des formations



Retrouvez nos formations dans leur dernière version :

- DBA1 : Administration PostgreSQL
  https://dali.bo/dba1
- DBA2 : Administration PostgreSQL avancé
  https://dali.bo/dba2
- DBA3 : Sauvegarde et réplication avec PostgreSQL
  https://dali.bo/dba3
- DEVPG : Développer avec PostgreSQL
  https://dali.bo/devpg
- PERF1 : PostgreSQL Performances
  https://dali.bo/perf1
- PERF2 : Indexation et SQL avancés
  https://dali.bo/perf2
- MIGORPG : Migrer d'Oracle à PostgreSQL
  https://dali.bo/migorpg
- HAPAT : Haute disponibilité avec PostgreSQL
  https://dali.bo/hapat

Les livres blancs

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

Téléchargement gratuit

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