Meetup Lille

pgBackRest Schrödinger's backups

28 janvier 2020

Stefan Fercot

pgBackRest Schrödinger's backups

Meetup Lille

TITRE : pgBackRest Schrödinger's backups
SOUS-TITRE : Meetup Lille

DATE: 28 janvier 2020

QUI SUIS-JE?

- Stefan Fercot
- aka. pgstef
- https://pgstef.github.io
- utilise PostgreSQL depuis 2010
- fan de pgBackRest
- @dalibo depuis 2017

DALIBO

- Services

Support	Formation	Conseil

- Participation active à la communauté
- https://www.dalibo.com/jobs

INTRODUCTION

- pgBackRest
  - c'est quoi ?
  - cas d'utilisation typique
- check_pgbackrest
  - que faut-il superviser et comment ?

pgBackRest est un outil super intéressant pour la gestion des sauvegardes et des restaurations, n'est-ce pas ? Sa commande "info" est très utile pour vérifier l'état des sauvegardes… mais…

Êtes-vous certains que les sauvegardes présentes correspondent bien à la politique de rétention définie ? Êtes-vous certains que tous les WAL archivés nécessaires à la restauration sont bien présents ?

`check_pgbackrest` est un outil pour vous aider à superviser cela.

Il vous permettra notamment de : * compter le nombre de sauvegardes "full" ; * vérifier l'âge de la sauvegarde la plus récente, ainsi que son type (complète, différentielle ou incrémentale) ; * vous assurer que tous les WAL archivés sont effectivement présents sur disque ; * …

Durant cette présentation, nous verrons quelques cas d'utilisation typique de pgBackRest et comment les superviser : * sauvegarde locale, * sur un hôte distant utilisant également pgBackRest, * sur un bucket S3.

Nous verrons comment utiliser l'outil manuellement ou comment l'intégrer dans un système de supervision compatible Nagios.

WRITE AHEAD LOG (WAL)

- Les transactions sont écrites séquentiellement dans le journal
  - et validées (`COMMIT`) une fois synchronisées sur disque
- Le rejeu des journaux de transactions (WAL) après un crash
  - remet la base dans un état cohérent

Les journaux de transactions (WAL) sont le mécanisme utilisé par PostgreSQL pour s'assurer qu'aucun changement validé (`COMMIT`) ne soit perdu. Les transactions sont écrites séquentiellement dans le journal de transaction et ces dernières sont considérées comme validées lorsque les changements sont écrits sur disque.

Ensuite, les processus en arrière-plan écrivent les changement dans les fichiers de données de façon asynchrone.

En cas de crash, les journaux de transactions sont rejoués pour remettre la base dans un état cohérent.

POINT-IN-TIME RECOVERY (PITR)

- Mise en œuvre combine
  - sauvegarde au niveau système de fichiers
  - archivage en continu des journaux de transactions (WAL)
- Restauration de la sauvegarde et rejeu des WAL archivés
- Il n'est pas nécessaire de rejouer l'entièreté des transactions inscrites dans les journaux

https://www.postgresql.org/docs/current/static/continuous-archiving.html

PGBACKREST

- Outil de gestion des sauvegardes et restaurations
- Depuis la version 2.21, réécrit entièrement en C
- Opère en local ou à distance (via SSH)
- Multi-processus
- Sauvegardes complète / différentielle / incrémentale
- Purge des sauvegardes et archives WAL inutiles
- Possibilité de chiffrement du dépôt
- ...

https://pgbackrest.org

MISE EN PLACE - ARCHIVAGE

```
# postgresql.conf
archive_mode = on
archive_command = 'pgbackrest --stanza=my_stanza archive-push %p'
```

INITIALISATION

```
$ pgbackrest --stanza=my_stanza stanza-create
P00   INFO: stanza-create command begin 2.23: ...
P00   INFO: stanza-create command end: completed successfully

$ pgbackrest --stanza=my_stanza check
P00   INFO: check command begin 2.23: ...
P00   INFO: WAL segment 000000010000000000000001 successfully archived to ...
P00   INFO: check command end: completed successfully
```

SAUVEGARDE COMPLÈTE

```
$ pgbackrest --stanza=my_stanza --type=full backup
P00   INFO: backup command begin 2.23: ...
P00   INFO: execute non-exclusive pg_start_backup() with label "...":
backup begins after the requested immediate checkpoint completes
P00   INFO: backup start archive = 000000010000000000000003, lsn = 0/3000060
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 = 000000010000000000000003, lsn = 0/3000138
P00   INFO: new backup label = 20200128-150158F
P00   INFO: backup command end: completed successfully
P00   INFO: expire command begin 2.23: ...
P00   INFO: expire command end: completed successfully
```

SAUVEGARDE DIFFÉRENTIELLE

```
$ pgbackrest --stanza=my_stanza --type=diff backup
P00   INFO: backup command begin 2.23: ...
P00   INFO: last backup label = 20200128-150158F, version = 2.23
P00   INFO: execute non-exclusive pg_start_backup() with label "...":
backup begins after the requested immediate checkpoint completes
P00   INFO: backup start archive = 000000010000000000000005, lsn = 0/5000028
P00   INFO: diff 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 = 0/5000138
P00   INFO: new backup label = 20200128-150158F_20200128-150245D
P00   INFO: backup command end: completed successfully
P00   INFO: expire command begin 2.23: ...
P00   INFO: expire command end: completed successfully
```

SAUVEGARDE INCRÉMENTALE

```
$ pgbackrest --stanza=my_stanza --type=incr backup
P00   INFO: backup command begin 2.23: ...
P00   INFO: last backup label = 20200128-150158F_20200128-150245D, version = 2.23
P00   INFO: execute non-exclusive pg_start_backup() with label "...":
backup begins after the requested immediate checkpoint completes
P00   INFO: backup start archive = 000000010000000000000007, lsn = 0/7000028
P00   INFO: incr 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 = 000000010000000000000007, lsn = 0/7000138
P00   INFO: new backup label = 20200128-150158F_20200128-150410I
P00   INFO: backup command end: completed successfully
P00   INFO: expire command begin 2.23: ...
P00   INFO: expire command end: completed successfully
```

COMMANDE INFO

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

db (current)
  wal archive min/max (12-1): 000000010000000000000003/000000010000000000000007

  full backup: 20200128-150158F
    timestamp start/stop: 2020-01-28 15:01:58 / 2020-01-28 15:02:14
    wal start/stop: 000000010000000000000003 / 000000010000000000000003
    database size: 24.2MB, backup size: 24.2MB
    repository size: 2.9MB, repository backup size: 2.9MB
...
```

CAS D'UTILISATION TYPIQUE

- Stockage local
- Stockage distant
- Stockage S3

STOCKAGE LOCAL (1)

```
# /etc/pgbackrest.conf
[global]
repo1-type=cifs
repo1-path=/var/lib/pgbackrest
repo1-retention-full=1
process-max=2
log-level-console=warn
log-level-file=info
start-fast=y
delta=y

[my_stanza]
pg1-path=/var/lib/pgsql/12/data
```

```
--REPO-TYPE

Repository Type Option (--repo-type)
Type of storage used for the repository.
The following repository types are supported:

    cifs - Like posix, but disables links and directory fsyncs
    posix - Posix-compliant file systems
    s3 - AWS Simple Storage Service
```

Attention au CIFS !

https://www.postgresql.org/docs/current/creating-cluster.html#CREATING-CLUSTER-NFS

STOCKAGE DISTANT (2)

- **pgsql-srv**

```
[global]
repo1-host=backup-srv
repo1-host-user=postgres
...
[my_stanza]
pg1-path=/var/lib/pgsql/12/data
```

- **backup-srv**

```
[global]
repo1-path=/var/lib/pgbackrest
repo1-retention-full=1
...
[my_stanza]
pg1-host=pgsql-srv
pg1-path=/var/lib/pgsql/12/data
```

EXÉCUTION DES COMMANDES EN STOCKAGE DISTANT

- **pgsql-srv**
  - `archive_command`
  - restore
- **backup-srv**
  - backup

STOCKAGE S3 (3)

```
[global]
repo1-path=/repo1
repo1-type=s3
repo1-s3-endpoint=minio.local
repo1-s3-bucket=pgbackrest
repo1-s3-verify-tls=n
repo1-s3-key=accessKey
repo1-s3-key-secret=***
repo1-s3-region=eu-west-3
```

```
...
[my_stanza]
pg1-path=/var/lib/pgsql/12/data
```

Configuration adaptée pour un scénario de test implémentant MinIO pour simuler un stockage compatible Amazon S3.

- `repo1` est un répertoire à l'intérieur du bucket `pgbackrest` ;
- `repo1-s3-verify-tls` est désactivé pour notre scénario de test.

CHECK_PGBACKREST

https://github.com/dalibo/check_pgbackrest

INSTALLATION

```
$ sudo yum install nagios-plugins-pgbackrest-1.7-1.noarch.rpm
#==============================================================================
Package                                  Repository
#==============================================================================
Installing:
 nagios-plugins-pgbackrest               nagios-plugins-pgbackrest-1.7-1.noarch

Installing for dependencies:
 nagios-common                           epel
 nagios-plugins                          epel
 perl-IO-Tty                             base
 perl-JSON                               base
 perl-Net-SFTP-Foreign                   epel
 perl-Sort-Key                           epel
#==============================================================================
```

```
$ /usr/lib64/nagios/plugins/check_pgbackrest --version
check_pgbackrest version 1.7, Perl 5.16.3
```

Attention, il faut que `epel-release` soit également installé.

https://github.com/dalibo/check_pgbackrest/releases

SERVICES DISPONIBLES

```
$ check_pgbackrest --list

List of available services:
  archives          Check WAL archives.
  check_pgb_version Check the version of this check_pgbackrest script.
  retention         Check the retention policy.
```

RÉTENTION

- Échoue quand
  - le nombre de sauvegardes complètes est inférieur à `--retention-full`
  - la dernière sauvegarde est plus ancienne que `--retention-age`
  - la dernière sauvegarde **complète** est plus ancienne que `--retention-age-to-full`

```
retention
  Fail when the number of full backups is less than the --retention-full argument.
  Fail when the newest backup is older than the --retention-age argument.
  Fail when the newest full backup is older than the --retention-age-to-full argument.
```

`--RETENTION-FULL`

```
$ check_pgbackrest --stanza=my_stanza
  --service=retention --retention-full=1

BACKUPS_RETENTION OK - backups policy checks ok |
  full=1 diff=1 incr=1
  latest=incr,20200128-150158F_20200128-150410I latest_age=120s
```

`--OUTPUT=HUMAN`

```
$ check_pgbackrest --stanza=my_stanza
  --service=retention --retention-full=1 --output=human

Service        : BACKUPS_RETENTION
Returns        : 0 (OK)
Message        : backups policy checks ok
Long message   : full=1
Long message   : diff=1
Long message   : incr=1
Long message   : latest=incr,20200128-150158F_20200128-150410I
Long message   : latest_age=2m24s
```

PLUSIEURS ARGUMENTS ENSEMBLE

```
$ check_pgbackrest --stanza=my_stanza
  --service=retention --retention-full=1 --output=human
  --retention-age=24h --retention-age-to-full=7d

Service         : BACKUPS_RETENTION
Returns         : 0 (OK)
Message         : backups policy checks ok
Long message    : full=1
Long message    : diff=1
Long message    : incr=1
Long message    : latest=incr,20200128-150158F_20200128-150410I
Long message    : latest_age=2m47s
Long message    : latest_full=20200128-150158F
Long message    : latest_full_age=5m
```

PEU IMPORTE LA LOCALISATION DES SAUVEGARDES ?

Se base uniquement sur la sortie de la commande `pgbackrest info` !

VÉRIFICATION DES ARCHIVES

- La commande `pgbackrest info`
  - affiche l'archive la plus ancienne (min) et la plus récente (max)
  - ne vérifie pas que la chaîne d'archive est bien complète sur disque
  - ne fourni pas l'age de l'archive la plus récente
  - ...

COMMENT ?

- 000000010000000200000003
  - 00000001 : timeline
  - 00000002 : wal
  - 00000003 : segment
- initdb `--wal-segsize=size`
  - à partir de la v11
  - par défaut 16MB
  - 256 segments par wal (>= v9.3)

Vérifier segment par segment, wal après wal !

Depuis la version 9.3, les segments vont de 00000000 à 000000FF. Précédemment, s'arrêtait à 000000FE.

SAUT DE TIMELINE

```
# 00000002.history
1 0/9000000 no recovery target specified
```

- Segment concerné : 000000010000000000000009

EXEMPLE : OOPS (1)

```
$ createdb bench
$ pgbench -i -s 100 bench
$ pgbackrest info --stanza=my_stanza
stanza: my_stanza
status: ok
cipher: none

db (current)
  wal archive min/max (12-1): 000000010000000000000003/000000010000000000000004D
...
```

STOCKAGE LOCAL (1)

```
$ check_pgbackrest --stanza=my_stanza
  --service=archives --repo-path=/var/lib/pgbackrest/archive

WAL_ARCHIVES OK - 81 WAL archived, latest archived since 1m27s |
  latest_archive_age=87s num_archives=81
```

```
--OUTPUT=HUMAN

$ check_pgbackrest --stanza=my_stanza
  --service=archives --repo-path=/var/lib/pgbackrest/archive --output=human
Service        : WAL_ARCHIVES
Returns        : 0 (OK)
Message        : 81 WAL archived
Message        : latest archived since 1m59s
Long message   : latest_archive_age=1m59s
Long message   : num_archives=81
Long message   : archives_dir=/var/lib/pgbackrest/archive/my_stanza/12-1
Long message   : min_wal=000000010000000000000003
Long message   : max_wal=000000010000000000000053
Long message   : oldest_archive=000000010000000000000003
Long message   : latest_archive=000000010000000000000053
Long message   : latest_bck_archive_start=000000010000000000000007
Long message   : latest_bck_type=incr
```

Oops (2)

```
$ rm -rf [...]/archive/my_stanza/12-1/0000000100000000/000000010000000000000001*
$ pgbackrest info --stanza=my_stanza
stanza: my_stanza
status: ok
cipher: none

db (current)
  wal archive min/max (12-1): 000000010000000000000003/000000010000000000000053
...
```

pgBackRest ne remonte pas d'erreur !

OOPS (3)

```
$ check_pgbackrest --stanza=my_stanza
  --service=archives --repo-path=/var/lib/pgbackrest/archive --output=human
Service        : WAL_ARCHIVES
Returns        : 2 (CRITICAL)
Message        : wrong sequence or missing file @ '000000010000000000000010'
...
Message        : wrong sequence or missing file @ '00000001000000000000001F'
...
Long message   : min_wal=000000010000000000000003
Long message   : max_wal=000000010000000000000053
Long message   : oldest_archive=000000010000000000000003
Long message   : latest_archive=000000010000000000000053
Long message   : latest_bck_archive_start=000000010000000000000007
Long message   : latest_bck_type=incr
```

- WARNING si archive manquante < `latest_bck_archive_start`
  - CRITICAL sinon

STOCKAGE DISTANT (2)

- depuis **pgsql-srv**
  - considérer stockage distant `--repo-host` et `--repo-host-user`
  - `Net::SFTP::Foreign`
- depuis **backup-srv**
  - considérer stockage local (voir précédemment)

https://metacpan.org/pod/Net::SFTP::Foreign

`--REPO-HOST`, `--REPO-HOST-USER`

```
$ check_pgbackrest --stanza=my_stanza
  --service=archives --repo-path=/var/lib/pgbackrest/archive
  --repo-host="backup-srv" --repo-host-user=postgres

WAL_ARCHIVES OK - 4 WAL archived, latest archived since 25m30s |
  latest_archive_age=25m30s num_archives=4
```

STOCKAGE S3 (3)

- Récupération de `repo1-s3-key` et `repo1-s3-key-secret`
  - depuis la configuration de pgBackRest
  - `Config::IniFiles`
- Connexion via API `Net::Amazon::S3`

- https://metacpan.org/pod/Config::IniFiles
- https://metacpan.org/pod/Net::Amazon::S3

`--REPO-S3`, `--REPO-S3-OVER-HTTP`

```
$ check_pgbackrest --stanza=my_stanza
  --service=archives --repo-path=/repo1/archive
  --repo-s3 --repo-s3-over-http

WAL_ARCHIVES OK - 4 WAL archived, latest archived since 1m7s |
  latest_archive_age=1m7s num_archives=4
```

ALLER PLUS LOIN

- `--ignore-archived-before`
  - ignore les archives générées **avant** cet intervalle
  - utile pour ne vérifier que les dernières archives
- `--ignore-archived-after`
  - ignore les archives générées **après** cet intervalle
  - utile sous de fortes charges d'archivage
- `--latest-archive-age-alert`
  - défini l'age maximum de la dernière archive avant alerte

```
Use the --ignore-archived-before argument to ignore the archived WALs generated before
interval. Used to only check the latest archives.

Use the --ignore-archived-after argument to ignore the archived WALs generated after t

The --latest-archive-age-alert argument defines the max age of the latest archived WAL
before raising a critical alert.
```

NAGIOS & DÉRIVÉS

- Format de sortie compatible avec la plupart des systèmes de supervision
  - Nagios
  - Naemon
  - Icinga
  - ...

RÉTENTION (STOCKAGE LOCAL)

```
object CheckCommand "by_ssh_pgbackrest_retention" {
  import "by_ssh"
  vars.by_ssh_command = "/usr/lib64/nagios/plugins/check_pgbackrest
  --stanza=\$stanza$ --service=retention
  --retention-full=\$retention_full$ --prefix='\$prefix$'"
}

object Service "pgbackrest_retention" {
  import "generic-service"
  host_name = "pgsql-srv"
  check_command = "by_ssh_pgbackrest_retention"
  vars.by_ssh_logname = "accessed_by_ssh"

  vars.stanza = "my_stanza"
  vars.retention_full = 1
  vars.prefix = "sudo -u postgres"
}
```

ARCHIVES (STOCKAGE LOCAL)

```
object CheckCommand "by_ssh_pgbackrest_archives" {
  import "by_ssh"
  vars.by_ssh_command = "/usr/lib64/nagios/plugins/check_pgbackrest
  --stanza=\$stanza$ --service=archives
  --repo-path=\$repo_path$ --prefix='\$prefix$'"
}

object Service "pgbackrest_archives" {
  import "generic-service"
```

```
  host_name = "pgsql-srv"
  check_command = "by_ssh_pgbackrest_archives"
  vars.by_ssh_logname = "accessed_by_ssh"

  vars.stanza = "my_stanza"
  vars.repo_path = "/var/lib/pgbackrest/archive"
  vars.prefix = "sudo -u postgres"
}
```

TESTS GRÂCE À VAGRANT

```
[check_pgbackrest]$ ls test
Makefile  perf  provision  README.md  regress  ssh  Vagrantfile
```

- vm.box : CentOS 7
- vm.provider : libvirt

SCÉNARIO 1

- pgBackRest configuré pour sauvegarder et archiver localement sur un point de montage CIFS
- `icinga-srv` exécute les commandes `check_pgbackrest` en SSH avec Icinga 2
- `pgsql-srv` héberge l'instance PostgreSQL à sauvegarder avec pgBackRest
- `backup-srv` héberge le partage CIFS

SCÉNARIO 2

- pgBackRest configuré pour sauvegarder et archiver à distance
- `icinga-srv` exécute les commandes `check_pgbackrest` en SSH avec Icinga 2
- `pgsql-srv` héberge l'instance PostgreSQL à sauvegarder avec pgBackRest
- `backup-srv` héberge les sauvegardes et archives de pgBackRest
- Les sauvegardes pgBackRest sont utilisées pour construire une instance secondaire en réplication sur `backup-srv`

SCÉNARIO 3

- pgBackRest configuré pour sauvegarder et archiver sur un bucket MinIO S3
- `icinga-srv` exécute les commandes `check_pgbackrest` en SSH avec Icinga 2
- `pgsql-srv` héberge l'instance PostgreSQL à sauvegarder avec pgBackRest
- `backup-srv` héberge le serveur MinIO

ÉVOLUTION (1)

- Utilisation de `pgbackrest ls`
  - pour récupérer la liste des archives
  - `mtime` disponible à partir de la version 2.21
  - en attente d'une commande `cat` pour les `.history`

```
$ pgbackrest help ls
pgBackRest 2.23 - 'ls' command help

List paths/files in the repository.

This is intended to be a general purpose list function, but for now it only
works on the repository.

Command Options:

  --filter                        filter output with a regular expression
  --output                        output format [default=text]
  --recurse                       include all subpaths in output [default=n]
  --sort                          sort output ascending, descending, or none
                                  [default=asc]
```

ÉVOLUTION (2)

- Support de Debian
  - scénario de test spécifique
  - génération de paquet `.deb` ?

CONTRIBUTION

- Toute contribution est la bienvenue !
  - GitHub issues
  - Rapports de bugs, retour d'utilisation,…
- Exemples dans le CHANGELOG
  - *Fix bad behavior on CIFS mount (reported by `renesepp`)*
  - *Add Amazon s3 support for archives service (Andrew E. Bruno)*
  - …

CONCLUSION

- Utiliser des outils de sauvegarde
- Ne pas se fier aux rapports d'états de ces outils
- Tenter de restaurer ses sauvegardes et…
  - superviser !

DES QUESTIONS ?