Unleash the Power within pgBackRest

PG Day'21 Russia

Stefan FERCOT

9 July 2021

Who Am I?

- Stefan Fercot
- aka. pgstef
- https://pgstef.github.io
- PostgreSQL user since 2010
- pgBackRest fan & contributor
- Database Backup Architect @EDB

Agenda

- pgBackRest
  - basic functionalities reminder
  - various backup storage types
    - multi-repository feature
  - less common operations
    - interact with a standby server
    - asynchronous archiving
  - diagnostic
  - monitoring

pgBackRest

- aims to be a simple, reliable backup and restore system
- written in C (migration completed in 2019)
- current release: 2.34 (June 7, 2021)
- custom protocol
  - local or remote operation (via SSH)

Installation

- *Use the PGDG repository, Luke!*
  - yum / dnf / apt-get install pgbackrest

Configuration

- `/etc/pgbackrest.conf`, example:

```
[global]
repo1-path=/var/lib/pgsql/13/backups
repo1-retention-full=1
log-level-console=info

[my_stanza]
pg1-path=/var/lib/pgsql/13/data
```

- main configuration in the `[global]` part
- each PostgreSQL cluster to backup has its own configuration, called `stanza`

Setup - archiving

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

Initialization

```
$ pgbackrest --stanza=my_stanza stanza-create
P00   INFO: stanza-create command begin 2.34: ...
P00   INFO: stanza-create for stanza 'my_stanza' on repo1
P00   INFO: stanza-create command end: completed successfully

$ pgbackrest --stanza=my_stanza check
P00   INFO: check command begin 2.34: ...
P00   INFO: check repo1 configuration (primary)
P00   INFO: check repo1 archive for WAL (primary)
P00   INFO: WAL segment 000000010000000000000001 successfully archived to '...' on repo1
P00   INFO: check command end: completed successfully
```

Full backup

```
$ pgbackrest --stanza=my_stanza --type=full backup
P00   INFO: backup command begin 2.34: ...
P00   INFO: execute non-exclusive pg_start_backup():
backup begins after the next regular checkpoint completes
P00   INFO: backup start archive = 000000010000000000000003, lsn = 0/3000028
P00   INFO: full backup size = 23.1MB
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: check archive for segment(s) 000000010000000000000003:000000010000000000000003
P00   INFO: new backup label = 20210629-123356F
P00   INFO: backup command end: completed successfully
```

```
P00   INFO: expire command begin 2.34: ...
P00   INFO: repo1: 13-1 remove archive,
start = 000000010000000000000001, stop = 000000010000000000000002
P00   INFO: expire command end: completed successfully
```

Backup types

- full
  - all database cluster files will be copied
  - no dependencies on previous backups
- incr
  - incremental from the last successful backup
- diff
  - like an incremental backup but always based on the last **full** backup

INFO command

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

    db (current)
        wal archive min/max (13): 000000010000000000000003/000000010000000000000006

        full backup: 20210629-123356F
            timestamp start/stop: 2021-06-29 12:33:56 / 2021-06-29 12:34:04
            wal start/stop: 000000010000000000000003 / 000000010000000000000003
            database size: 23.1MB, database backup size: 23.1MB
            repo1: backup set size: 2.8MB, backup size: 2.8MB
...
```

Where do I store my backups?

Do not keep your backup storage on the PostgreSQL host!

- directly attached storage (`repo1-type`)
- dedicated remote host (`repo1-host`)

Repository storage types

- `repo1-type`
  - azure - Azure Blob Storage Service
  - cifs - Like posix, but disables links and directory fsyncs
  - gcs - Google Cloud Storage
  - posix - Posix-compliant file systems
  - s3 - AWS Simple Storage Service

Dedicated remote host

- install pgBackRest
- create a specific user on the backup server
- setup passwordless SSH connection between the hosts

Dedicated remote host - configuration

- PostgreSQL server

```
[global]
repo1-host=backup-srv
repo1-host-user=pgbackrest

[my_stanza]
pg1-path=/var/lib/pgsql/13/data
```

- Backup server

```
[global]
repo1-path=/backup_space

[my_stanza]
pg1-host=pgsql-srv
pg1-host-user=postgres
pg1-path=/var/lib/pgsql/13/data
```

Command execution with remote storage

- PostgreSQL server
  - `archive_command`
  - restore
- Backup server
  - backup

Using multiple repositories

- introduced in 2.33 (April 5, 2021)
  - redundancy
  - various retention settings
  - ...

```
# example
repo1-path=.../repo1
repo1-retention-full=2
repo2-path=.../repo2
repo2-retention-full=1
```

`--repo` option

- backward compatibility
  - not required when only `repo1` is configured
- when a single repository is configured
  - recommended to use `repo1` in the configuration

`stanza-create` command

- automatically operates on all configured repositories

```
$ pgbackrest --stanza=my_stanza stanza-create
P00   INFO: stanza-create command begin 2.34: ...
P00   INFO: stanza-create for stanza 'my_stanza' on repo1
P00   INFO: stanza-create for stanza 'my_stanza' on repo2
P00   INFO: stanza-create command end: completed successfully
```

EDB

`check` command

- triggers a new WAL segment to be archived
- tries to push it to all defined repositories

```
$ pgbackrest --stanza=my_stanza check
P00   INFO: check command begin 2.34: ...
P00   INFO: check repo1 configuration (primary)
P00   INFO: check repo2 configuration (primary)
P00   INFO: check repo1 archive for WAL (primary)
P00   INFO: WAL segment ... successfully archived to '...' on repo1
P00   INFO: check repo2 archive for WAL (primary)
P00   INFO: WAL segment ... successfully archived to '...' on repo2
P00   INFO: check command end: completed successfully
```

`archive-push` command

- tries to push the WAL archive to all reachable repositories
  - an error prevent PostgreSQL to remove/recycle the WAL file!
  - `archive-async=y` brings fault-tolerance

```
P00  DEBUG:     storage/storage::storageNewWrite: => {
  type: posix, name: {".../repo1/archive/my_stanza/13-1/0000000100000000/
              000000010000000000000008-097a6928789bb4e145ff19347a2353feabbf00f0.gz"},
...
P00  DEBUG:     storage/storage::storageNewWrite: => {
  type: posix, name: {".../repo2/archive/my_stanza/13-1/0000000100000000/
              000000010000000000000008-097a6928789bb4e145ff19347a2353feabbf00f0.gz"},
...
P00   INFO: pushed WAL file '000000010000000000000008' to the archive
```

Backups

- scheduled individually for each repository
- without `--repo`, `repo1` is used

```
$ pgbackrest backup --stanza=my_stanza --type=full
P00   INFO: backup command begin 2.34: ...
P00   INFO: repo option not specified, defaulting to repo1
P00   INFO: execute non-exclusive pg_start_backup():
backup begins after the next regular checkpoint completes
P00   INFO: backup start archive = 00000001000000000000000A, lsn = 0/A000028
P00   INFO: full backup size = 23.1MB
P00   INFO: execute non-exclusive pg_stop_backup() and wait for all WAL segments to archive
P00   INFO: backup stop archive = 00000001000000000000000A, lsn = 0/A000138
P00   INFO: check archive for segment(s) 00000001000000000000000A:00000001000000000000000A
P00   INFO: new backup label = 20210629-134343F
P00   INFO: backup command end: completed successfully
```

```
P00   INFO: expire command begin 2.34: ...
P00   INFO: repo1: 13-1 remove archive,
start = 000000010000000000000007, stop = 000000010000000000000009
P00   INFO: expire command end: completed successfully
```

Show information

- default order sorting backups by dates mixing the repositories
  - might be confusing to find the backups depending on each other

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

    db (current)
        wal archive min/max (13): 00000001000000000000000A/00000001000000000000000C

        full backup: 20210629-134343F
            timestamp start/stop: 2021-06-29 13:43:43 / 2021-06-29 13:43:52
            wal start/stop: 00000001000000000000000A / 00000001000000000000000A
            database size: 23.1MB, database backup size: 23.1MB
            repo1: backup set size: 2.8MB, backup size: 2.8MB

        full backup: 20210629-134519F
            timestamp start/stop: 2021-06-29 13:45:19 / 2021-06-29 13:45:29
            wal start/stop: 00000001000000000000000C / 00000001000000000000000C
            database size: 23.2MB, database backup size: 23.2MB
            repo2: backup set size: 2.8MB, backup size: 2.8MB
```

Show information per repository

```
$ pgbackrest info --stanza=my_stanza --repo=2
stanza: my_stanza
    status: ok
    cipher: none

    db (current)
        wal archive min/max (13): 00000001000000000000000C/00000001000000000000000C

        full backup: 20210629-134519F
            timestamp start/stop: 2021-06-29 13:45:19 / 2021-06-29 13:45:29
            wal start/stop: 00000001000000000000000C / 00000001000000000000000C
            database size: 23.2MB, database backup size: 23.2MB
            repo2: backup set size: 2.8MB, backup size: 2.8MB
```

Recovery

```
restore_command = 'pgbackrest --stanza=my_stanza archive-get %f "%p"'
```

- `archive-get` will look into the repositories in priority order
  - (`repo1` > `repo2` > ...)
- tolerate gaps!

Less common operations

- refresh Streaming Replication standby
- take backups from the standby server
- asynchronously push or get WAL segments
- selective restore

EDB

Refresh Streaming Replication standby

- repository reachable from both nodes
- add extra stanza configuration on the standby

```
recovery-option=primary_conninfo=host=primary user=replication_user
```

- perform a `delta` restore

```
$ pgbackrest --stanza=my_stanza --type=standby --delta restore
```

- check `primary_conninfo` and `restore_command` before restarting the service

EDB

Take backups from the standby server

- `backup-standby` option

```
[global]
...
backup-standby=y

[my_stanza]
pg1-path=/var/lib/pgsql/13/data
pg2-host=primary
pg2-path=/var/lib/pgsql/13/data
recovery-option=primary_conninfo=host=primary user=replication_user
```

- backup started on primary
  - wait replay location on standby
  - files are copied from the standby

EDB

`archive-push` WAL segments

- triggered by `archive_command`
- using `archive-async=y`
  - write temporary data (acknowledgments) into the `spool-path`
  - using `process-max` processes
- `archive-push-queue-max`
  - maximum size of the PostgreSQL archive queue
  - prevent the WAL space from filling up until PostgreSQL stops completely…
  - …but generate **missing archives**!
- very important to monitor archiving to ensure it continues working

EDB

Asynchronously get WAL segments

- `archive-get` using `archive-async=y`
  - using `process-max` processes
  - prefetch `archive-get-queue-max` amount of WAL segments to speed up recovery

Selective restore

- `--db-include`
  - databases not specifically included will be restored as sparse, zeroed files
  - built-in databases (template0, template1, and postgres) are always restored unless specifically excluded
- `--db-exclude`
  - databases excluded will be restored as sparse, zeroed files
  - with the `--db-include` option, only apply to built-in databases
- `DROP DATABASE` to remove the zeroed databases after recovery

Diagnostic

- checksums
- `check` command
- `verify` command

Checksums

- PostgreSQL `initdb --data-checksums`
  - PGSETUP_INITDB_OPTIONS
- pg_checksums
  - enable, disable or check data checksums **offline**
- pgBackRest `--checksum-page`
  - validate all data page checksums while backing up a cluster
  - automatically enabled when data page checksums are enabled on the cluster

`check` command

- validates configuration and `archive_command` setting
- calls
  - `pg_create_restore_point('pgBackRest Archive Check')`
  - and `pg_switch_wal()`

PostgreSQL archiving process

- `archive-push` output is sent to PostgreSQL logs

```
[global:archive-push]
log-level-console=debug
```

EDB

`verify` command

- internal command only, **work in progress**

```
pgBackRest 2.34 - 'verify' command help

Verify the contents of the repository.

Verify will attempt to determine if the backups and archives in the repository
are valid.
```

- WAL validation and backup files verification

```
INFO: Results:
  archiveId: 13-1, total WAL checked: 4, total valid WAL: 4
    missing: 0, checksum invalid: 0, size invalid: 0, other: 0
  backup: 20210629-123356F, status: valid, total files checked: 936, total valid files: 936
    missing: 0, checksum invalid: 0, size invalid: 0, other: 0
```

Monitoring

Schrödinger’s Law of Backups

The condition/state of any backup is unknown until a restore is attempted.

- play with `pgbackrest info --output=json` within PostgreSQL…
- … or use check_pgbackrest

check_pgbackrest

- whatever the backups location?
  - only based on `pgbackrest info` output and `repo` commands!

Available services

```
$ 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.
```

EDB

Retention

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

`--retention-full`

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

BACKUPS_RETENTION OK - backups policy checks ok |
  full=1 diff=0 incr=0 latest=full,20210629-123356F latest_age=781s
```

--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=0
Long message     : incr=0
Long message     : latest=full,20210629-123356F
Long message     : latest_age=14m4s
```

Multiple arguments together

```
$ 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=0
Long message    : incr=0
Long message    : latest=full,20210629-123356F
Long message    : latest_age=14m18s
Long message    : latest_full=20210629-123356F
Long message    : latest_full_age=14m18s
```

Archives

- `info` command
  - shows the oldest (min) archive and the most recent one (max)
  - doesn’t check if all the archives in between are really on the disk
  - …
- `verify` command is still experimental

Archives (2)

```
$ check_pgbackrest --stanza=my_stanza --service=archives

WAL_ARCHIVES OK - 4 WAL archived, latest archived since 3m21s |
  latest_archive_age=201s num_archives=4
```

`--output=human`

```
$ check_pgbackrest --stanza=my_stanza --service=archives --output=human

Service        : WAL_ARCHIVES
Returns        : 0 (OK)
Message        : 4 WAL archived
Message        : latest archived since 3m41s
Long message   : latest_archive_age=3m41s
Long message   : num_archives=4
...
Long message   : latest_archive=000000010000000000000006
Long message   : latest_bck_archive_start=000000010000000000000003
Long message   : latest_bck_type=full
...
```

EDB

Oops (1)

```
$ rm -rf [...]/archive/my_stanza/13-1/0000000100000000/000000010000000000000005-*
$ pgbackrest info --stanza=my_stanza
stanza: my_stanza
    status: ok
    cipher: none

    db (current)
        wal archive min/max (13): 000000010000000000000003/000000010000000000000006
...
```

```
$ pgbackrest verify --stanza=my_stanza
P00   INFO: Results:
  archiveId: 13-1, total WAL checked: 3, total valid WAL: 3
    missing: 0, checksum invalid: 0, size invalid: 0, other: 0
  backup: 20210629-123356F, status: valid, total files checked: 936, total valid files: 936
    missing: 0, checksum invalid: 0, size invalid: 0, other: 0
```

pgBackRest doesn’t report any error!

EDB

Oops (2)

```
$ check_pgbackrest --stanza=my_stanza --service=archives --output=human
Service         : WAL_ARCHIVES
Returns         : 2 (CRITICAL)
Message         : wrong sequence, 1 missing file(s) (...)
Long message    : latest_archive_age=5m49s
Long message    : num_archives=3
Long message    : num_missing_archives=1
Long message    : oldest_missing_archive=000000010000000000000005
Long message    : latest_missing_archive=000000010000000000000005
...
```

- WARNING if missing archive < `latest_bck_archive_start`
  - CRITICAL otherwise

EDB

Where

- official website: https://pgbackrest.org
- user guides: https://pgbackrest.org/user-guide.html
- code: https://github.com/pgbackrest/pgbackrest
- check_pgbackrest: https://github.com/pgstef/check_pgbackrest
- rpm and deb: in the PGDG repositories!

Conclusion

- pgBackRest is a powerful tool
  - with a lot of features and possibilities
- don’t forget *Schrödinger’s Law of Backups*
  - monitor backups and archiving system

Questions?

Thank you for your attention!