pgBackRest User Guide

Open Source PostgreSQL Backup and Restore Utility

Version 2.35

Crunchy Data Solutions, Inc.

September 8, 2021

Table of Contents

1 Introduction **2**

2 Concepts **3**
- 2.1 Backup . . . 3
- 2.2 Restore . . . 3
- 2.3 Write Ahead Log (WAL) . . . 3
- 2.4 Encryption . . . 4

3 Upgrading pgBackRest **5**
- 3.1 Upgrading pgBackRest from v1 to v2 . . . 5

4 Build **6**

5 Installation **7**

6 Quick Start **9**
- 6.1 Setup Demo Cluster . . . 9
- 6.2 Configure Cluster Stanza . . . 9
- 6.3 Create the Repository . . . 11
- 6.4 Configure Archiving . . . 12
- 6.5 Configure Retention . . . 13
- 6.6 Configure Repository Encryption . . . 13
- 6.7 Create the Stanza . . . 14
- 6.8 Check the Configuration . . . 14
- 6.9 Perform a Backup . . . 15
- 6.10 Schedule a Backup . . . 17
- 6.11 Backup Information . . . 17
- 6.12 Restore a Backup . . . 19

7 Monitoring **20**
- 7.1 In PostgreSQL . . . 20

8 Retention **22**
- 8.1 Full Backup Retention . . . 22
- 8.2 Differential Backup Retention . . . 24
- 8.3 Archive Retention . . . 25

9 Restore **27**
- 9.1 File Ownership . . . 27
- 9.2 Delta Option . . . 27
- 9.3 Restore Selected Databases . . . 28

10 Point-in-Time Recovery **32**

11 Multiple Repositories **38**

12 AZURE-COMPATIBLE OBJECT STORE SUPPORT 39

13 S3-COMPATIBLE OBJECT STORE SUPPORT 41

14 GCS-COMPATIBLE OBJECT STORE SUPPORT 45

15 DELETE A STANZA 47

16 DEDICATED REPOSITORY HOST 49
16.1 INSTALLATION 49
16.2 SETUP PASSWORDLESS SSH 50
16.3 CONFIGURATION 51
16.4 PERFORM A BACKUP 52
16.5 RESTORE A BACKUP 52

17 PARALLEL BACKUP / RESTORE 53

18 STARTING AND STOPPING 55

19 REPLICATION 57
19.1 INSTALLATION 57
19.2 SETUP PASSWORDLESS SSH 57
19.3 HOT STANDBY 58
19.4 STREAMING REPLICATION 61

20 ASYNCHRONOUS ARCHIVING 65
20.1 ARCHIVE PUSH 66
20.2 ARCHIVE GET 68

21 BACKUP FROM A STANDBY 70

22 UPGRADING POSTGRESQL 72

1 Introduction

This user guide is intended to be followed sequentially from beginning to end — each section depends on the last. For example, the RESTORE section relies on setup that is performed in the QUICK START section. Once pgBackRest is up and running then skipping around is possible but following the user guide in order is recommended the first time through.

Although the examples are targeted at RHEL/CentOS 7-8 and PostgreSQL 9.6-11, it should be fairly easy to apply this guide to any Unix distribution and PostgreSQL version. The only OS-specific commands are those to create, start, stop, and drop PostgreSQL clusters. The pgBackRest commands will be the same on any Unix system though the location to install the executable may vary.

Configuration information and documentation for PostgreSQL can be found in the PostgreSQL MANUAL.

A somewhat novel approach is taken to documentation in this user guide. Each command is run on a virtual machine when the documentation is built from the XML source. This means you can have a high confidence that the commands work correctly in the order presented. Output is captured and displayed below the command when appropriate. If the output is not included it is because it was deemed not relevant or was considered a distraction from the narrative.

All commands are intended to be run as an unprivileged user that has sudo privileges for both the `root` and `postgres` users. It's also possible to run the commands directly as their respective users without modification and in that case the `sudo` commands can be stripped off.

2 Concepts

The following concepts are defined as they are relevant to pgBackRest, PostgreSQL, and this user guide.

2.1 Backup

A backup is a consistent copy of a database cluster that can be restored to recover from a hardware failure, to perform Point-In-Time Recovery, or to bring up a new standby.

Full Backup: pgBackRest copies the entire contents of the database cluster to the backup. The first backup of the database cluster is always a Full Backup. pgBackRest is always able to restore a full backup directly. The full backup does not depend on any files outside of the full backup for consistency.

Differential Backup: pgBackRest copies only those database cluster files that have changed since the last full backup. pgBackRest restores a differential backup by copying all of the files in the chosen differential backup and the appropriate unchanged files from the previous full backup. The advantage of a differential backup is that it requires less disk space than a full backup, however, the differential backup and the full backup must both be valid to restore the differential backup.

Incremental Backup: pgBackRest copies only those database cluster files that have changed since the last backup (which can be another incremental backup, a differential backup, or a full backup). As an incremental backup only includes those files changed since the prior backup, they are generally much smaller than full or differential backups. As with the differential backup, the incremental backup depends on other backups to be valid to restore the incremental backup. Since the incremental backup includes only those files since the last backup, all prior incremental backups back to the prior differential, the prior differential backup, and the prior full backup must all be valid to perform a restore of the incremental backup. If no differential backup exists then all prior incremental backups back to the prior full backup, which must exist, and the full backup itself must be valid to restore the incremental backup.

2.2 Restore

A restore is the act of copying a backup to a system where it will be started as a live database cluster. A restore requires the backup files and one or more WAL segments in order to work correctly.

2.3 Write Ahead Log (WAL)

WAL is the mechanism that PostgreSQL uses to ensure that no committed changes are lost. Transactions are written sequentially to the WAL and a transaction is considered to be committed when those writes are flushed to disk. Afterwards, a background process writes the changes into the main database cluster files (also known as the heap). In the event of a crash, the WAL is replayed to make the database consistent.

WAL is conceptually infinite but in practice is broken up into individual 16MB files called segments. WAL segments follow the naming convention `0000000100000A1E000000FE` where the first 8 hexadecimal digits represent the timeline and the next 16 digits are the logical sequence number (LSN).

2.4 Encryption

Encryption is the process of converting data into a format that is unrecognizable unless the appropriate password (also referred to as passphrase) is provided.

pgBackRest will encrypt the repository based on a user-provided password, thereby preventing unauthorized access to data stored within the repository.

3 Upgrading pgBackRest

3.1 Upgrading pgBackRest from v1 to v2

Upgrading from `v1` to `v2` is fairly straight-forward. The repository format has not changed and all non-deprecated options from `v1` are accepted, so for most installations it is simply a matter of installing the new version.

However, there are a few caveats:

- The deprecated `thread-max` option is no longer valid. Use `process-max` instead.
- The deprecated `archive-max-mb` option is no longer valid. This has been replaced with the `archive-push-queue-max` option which has different semantics.
- The default for the `backup-user` option has changed from `backrest` to `pgbackrest`.
- In `v2.02` the default location of the pgBackRest configuration file has changed from `/etc/pgbackrest.` to `/etc/pgbackrest/pgbackrest.conf`. If `/etc/pgbackrest/pgbackrest.conf` does not exist, the `/etc/pgbackrest.conf` file will be loaded instead, if it exists.

Many option names have changed to improve consistency although the old names from `v1` are still accepted. In general, `db-*` options have been renamed to `pg-*` and `backup-*`/`retention-*` options have been renamed to `repo-*` when appropriate.

PostgreSQL and repository options must be indexed when using the new names introduced in `v2`, e.g. `pg1-host`, `pg1-path`, `repo1-path`, `repo1-type`, etc. Only one repository is allowed currently but more flexibility is planned for `v2`.

4 Build

RHEL/CentOS 7-8 packages for pgBackRest are available from CRUNCHY DATA or YUM.POSTGRESQL.ORG, but it is also easy to download the source and install manually.

When building from source it is best to use a build host rather than building on production. Many of the tools required for the build should generally not be installed in production. pgBackRest consists of a single executable so it is easy to copy to a new host once it is built.

build — *Download version* `2.35` *of pgBackRest to* `/build` *path*

```
mkdir -p /build
wget -q -O - \
   https://github.com/pgbackrest/pgbackrest/archive/release/2.35.tar.gz | \
   tar zx -C /build
```

build — *Install build dependencies*

```
sudo yum install make gcc postgresql10-devel \
   openssl-devel libxml2-devel lz4-devel libzstd-devel bzip2-devel libyaml-devel
```

build — *Configure and compile pgBackRest*

```
cd /build/pgbackrest-release-2.35/src && ./configure && make
```

5 Installation

A new host named **pg-primary** is created to contain the demo cluster and run pgBackRest examples.

pgBackRest needs to be installed from a package or installed manually as shown here.

pg-primary — *Install dependencies*

```
sudo yum install postgresql-libs
```

pg-primary — *Copy pgBackRest binary from build host*

```
sudo scp build:/build/pgbackrest-release-2.35/src/pgbackrest /usr/bin
sudo chmod 755 /usr/bin/pgbackrest
```

pgBackRest requires log and configuration directories and a configuration file.

pg-primary — *Create pgBackRest configuration file and directories*

```
sudo mkdir -p -m 770 /var/log/pgbackrest
sudo chown postgres:postgres /var/log/pgbackrest
sudo mkdir -p /etc/pgbackrest
sudo mkdir -p /etc/pgbackrest/conf.d
sudo touch /etc/pgbackrest/pgbackrest.conf
sudo chmod 640 /etc/pgbackrest/pgbackrest.conf
sudo chown postgres:postgres /etc/pgbackrest/pgbackrest.conf
```

pgBackRest should now be properly installed but it is best to check. If any dependencies were missed then you will get an error when running pgBackRest from the command line.

pg-primary — *Make sure the installation worked*

```
sudo -u postgres pgbackrest

Output:

pgBackRest 2.35 - General help

Usage:
   pgbackrest [options] [command]

Commands:
   archive-get Get a WAL segment from the archive.
   archive-push Push a WAL segment to the archive.
   backup      Backup a database cluster.
   check       Check the configuration.
   expire      Expire backups that exceed retention.
   help        Get help.
   info        Retrieve information about backups.
   repo-get    Get a file from a repository.
   repo-ls     List files in a repository.
```

```
   restore     Restore a database cluster.
   stanza-create Create the required stanza data.
   stanza-delete Delete a stanza.
   stanza-upgrade Upgrade a stanza.
   start       Allow pgBackRest processes to run.
   stop        Stop pgBackRest processes from running.
   version     Get version.

Use 'pgbackrest help [command]' for more information.
```

6 Quick Start

The Quick Start section will cover basic configuration of pgBackRest and PostgreSQL and introduce the `backup`, `restore`, and `info` commands.

6.1 Setup Demo Cluster

Creating the demo cluster is optional but is strongly recommended, especially for new users, since the example commands in the user guide reference the demo cluster; the examples assume the demo cluster is running on the default port (i.e. 5432). The cluster will not be started until a later section because there is still some configuration to do.

pg-primary — *Create the demo cluster*

```
sudo -u postgres /usr/pgsql-10/bin/initdb \
   -D /var/lib/pgsql/10/data -k -A peer
```

By default PostgreSQL will only accept local connections. The examples in this guide will require connections from other servers so `listen_addresses` is configured to listen on all interfaces. This may not be appropriate for secure installations.

pg-primary*:/var/lib/pgsql/10/data/postgresql.conf* — *Set* `listen_addresses`

```
listen_addresses = '*'
```

For demonstration purposes the `log_line_prefix` setting will be minimally configured. This keeps the log output as brief as possible to better illustrate important information.

pg-primary*:/var/lib/pgsql/10/data/postgresql.conf* — *Set* `log_line_prefix`

```
listen_addresses = '*'
log_line_prefix = ''
```

By default RHEL/CentOS 7-8 includes the day of the week in the log filename. This makes automating the user guide a bit more complicated so the `log_filename` is set to a constant.

pg-primary*:/var/lib/pgsql/10/data/postgresql.conf* — *Set* `log_filename`

```
listen_addresses = '*'
log_filename = 'postgresql.log'
log_line_prefix = ''
```

6.2 Configure Cluster Stanza

A stanza is the configuration for a PostgreSQL database cluster that defines where it is located, how it will be backed up, archiving options, etc. Most db servers will only have one Postgres database cluster

and therefore one stanza, whereas backup servers will have a stanza for every database cluster that needs to be backed up.

It is tempting to name the stanza after the primary cluster but a better name describes the databases contained in the cluster. Because the stanza name will be used for the primary and all replicas it is more appropriate to choose a name that describes the actual function of the cluster, such as app or dw, rather than the local cluster name, such as main or prod.

The name 'demo' describes the purpose of this cluster accurately so that will also make a good stanza name.

pgBackRest needs to know where the base data directory for the PostgreSQL cluster is located. The path can be requested from PostgreSQL directly but in a recovery scenario the PostgreSQL process will not be available. During backups the value supplied to pgBackRest will be compared against the path that PostgreSQL is running on and they must be equal or the backup will return an error. Make sure that `pg-path` is exactly equal to `data_directory` in `postgresql.conf`.

By default RHEL/CentOS 7-8 stores clusters in `/var/lib/pgsql/[version]/data` so it is easy to determine the correct path for the data directory.

When creating the `/etc/pgbackrest/pgbackrest.conf` file, the database owner (usually `postgres`) must be granted read privileges.

pg-primary*:/etc/pgbackrest/pgbackrest.conf — Configure the PostgreSQL cluster data directory*

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

pgBackRest configuration files follow the Windows INI convention. Sections are denoted by text in brackets and key/value pairs are contained in each section. Lines beginning with `#` are ignored and can be used as comments.

There are multiple ways the pgBackRest configuration files can be loaded:

- `config` and `config-include-path` are default: the default config file will be loaded, if it exists, and `*.conf` files in the default config include path will be appended, if they exist.
- `config` option is specified: only the specified config file will be loaded and is expected to exist.
- `config-include-path` is specified: `*.conf` files in the config include path will be loaded and the path is required to exist. The default config file will be be loaded if it exists. If it is desirable to load only the files in the specified config include path, then the `--no-config` option can also be passed.
- `config` and `config-include-path` are specified: using the user-specified values, the config file will be loaded and `*.conf` files in the config include path will be appended. The files are expected to exist.
- `config-path` is specified: this setting will override the base path for the default location of the config file and/or the base path of the default config-include-path setting unless the config and/or config-include-path option is explicitly set.

The files are concatenated as if they were one big file; order doesn't matter, but there is precedence based on sections. The precedence (highest to lowest) is:

- [*stanza*:*command*]
- [*stanza*]
- [global:*command*]
- [global]

NOTE: `--config`, `--config-include-path` *and* `--config-path` *are command-line only options.*

pgBackRest can also be configured using environment variables as described in the COMMAND REFERENCE.

pg-primary — *Configure* `log-path` *using the environment*

```
sudo -u postgres bash -c ' \
   export PGBACKREST_LOG_PATH=/path/set/by/env && \
   pgbackrest --log-level-console=error help backup log-path'

Output:

pgBackRest 2.35 - 'backup' command - 'log-path' option help

Path where log files are stored.

The log path provides a location for pgBackRest to store log files. Note that
if log-level-file=off then no log path is required.

current: /path/set/by/env
default: /var/log/pgbackrest
```

6.3 Create the Repository

The repository is where pgBackRest stores backups and archives WAL segments.

It may be difficult to estimate in advance how much space you'll need. The best thing to do is take some backups then record the size of different types of backups (full/incr/diff) and measure the amount of WAL generated per day. This will give you a general idea of how much space you'll need, though of course requirements will likely change over time as your database evolves.

For this demonstration the repository will be stored on the same host as the PostgreSQL server. This is the simplest configuration and is useful in cases where traditional backup software is employed to backup the database host.

pg-primary — *Create the pgBackRest repository*

```
sudo mkdir -p /var/lib/pgbackrest
```

```
sudo chmod 750 /var/lib/pgbackrest
sudo chown postgres:postgres /var/lib/pgbackrest
```

The repository path must be configured so pgBackRest knows where to find it.

pg-primary:`/etc/pgbackrest/pgbackrest.conf` — *Configure the pgBackRest repository path*

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

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

Multiple repositories may also be configured. See MULTIPLE REPOSITORIES for details.

6.4 Configure Archiving

Backing up a running PostgreSQL cluster requires WAL archiving to be enabled. Note that *at least* one WAL segment will be created during the backup process even if no explicit writes are made to the cluster.

pg-primary:`/var/lib/pgsql/10/data/postgresql.conf` — *Configure archive settings*

```
archive_command = 'pgbackrest --stanza=demo archive-push %p'
archive_mode = on
listen_addresses = '*'
log_filename = 'postgresql.log'
log_line_prefix = ''
max_wal_senders = 3
wal_level = replica
```

`%p` is how PostgreSQL specifies the location of the WAL segment to be archived. Setting `wal_level` to at least `replica` and increasing `max_wal_senders` is a good idea even if there are currently no replicas as this will allow them to be added later without restarting the primary cluster.

The PostgreSQL cluster must be restarted after making these changes and before performing a backup.

pg-primary — *Restart the demo cluster*

```
sudo systemctl restart postgresql-10.service
```

When archiving a WAL segment is expected to take more than 60 seconds (the default) to reach the pgBackRest repository, then the pgBackRest `archive-timeout` option should be increased. Note that this option is not the same as the PostgreSQL `archive_timeout` option which is used to force a WAL segment switch; useful for databases where there are long periods of inactivity. For more information on the PostgreSQL `archive_timeout` option, see PostgreSQL WRITE AHEAD LOG.

The `archive-push` command can be configured with its own options. For example, a lower compression level may be set to speed archiving without affecting the compression used for backups.

pg-primary:/etc/pgbackrest/pgbackrest.conf — *Config* archive-push *to use a lower compression level*

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

[global]
repo1-path=/var/lib/pgbackrest

[global:archive-push]
compress-level=3
```

This configuration technique can be used for any command and can even target a specific stanza, e.g. demo:archive-push.

6.5 Configure Retention

pgBackRest expires backups based on retention options.

pg-primary:/etc/pgbackrest/pgbackrest.conf — *Configure retention to 2 full backups*

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

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

[global:archive-push]
compress-level=3
```

More information about retention can be found in the RETENTION section.

6.6 Configure Repository Encryption

The repository will be configured with a cipher type and key to demonstrate encryption. Encryption is always performed client-side even if the repository type (e.g. S3 or other object store) supports encryption.

It is important to use a long, random passphrase for the cipher key. A good way to generate one is to run: openssl rand -base64 48.

pg-primary:/etc/pgbackrest/pgbackrest.conf — *Configure pgBackRest repository encryption*

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

```
[global]
repo1-cipher-pass=zWaf6XtpjIVZC5444yXB+cgFDFl7MxGlgkZSaoPvTGirhPygu4jOKOXf9LO4vjfO
repo1-cipher-type=aes-256-cbc
repo1-path=/var/lib/pgbackrest
repo1-retention-full=2

[global:archive-push]
compress-level=3
```

Once the repository has been configured and the stanza created and checked, the repository encryption settings cannot be changed.

6.7 Create the Stanza

The `stanza-create` command must be run to initialize the stanza. It is recommended that the `check` command be run after `stanza-create` to ensure archiving and backups are properly configured.

pg-primary — *Create the stanza and check the configuration*

```
sudo -u postgres pgbackrest --stanza=demo --log-level-console=info stanza-create

Output:

P00  INFO: stanza-create command begin 2.35: --exec-id=736-15962963 --log-level-
   console=info --log-level-stderr=off --no-log-timestamp --pg1-path=/var/lib/
   pgsql/10/data --repo1-cipher-pass=<redacted> --repo1-cipher-type=aes-256-cbc --
   repo1-path=/var/lib/pgbackrest --stanza=demo
P00  INFO: stanza-create for stanza 'demo' on repo1
P00  INFO: stanza-create command end: completed successfully
```

6.8 Check the Configuration

The `check` command validates that pgBackRest and the `archive_command` setting are configured correctly for archiving and backups for the specified stanza. It will attempt to check all repositories and databases that are configured for the host on which the command is run. It detects misconfigurations, particularly in archiving, that result in incomplete backups because required WAL segments did not reach the archive. The command can be run on the PostgreSQL or repository host. The command may also be run on the standby host, however, since `pg_switch_xlog()`/`pg_switch_wal()` cannot be performed on the standby, the command will only test the repository configuration.

Note that `pg_create_restore_point('pgBackRest Archive Check')` and `pg_switch_xlog()`/`pg_switch_wal()` are called to force PostgreSQL to archive a WAL segment. Restore points are only supported in PostgreSQL $\geq$ 9.1 so for older versions the `check`

command may fail if there has been no write activity since the last log rotation, therefore it is recommended that activity be generated by the user if there have been no writes since the last WAL switch before running the `check` command.

pg-primary — *Check the configuration*

```
sudo -u postgres pgbackrest --stanza=demo --log-level-console=info check

Output:

P00  INFO: check command begin 2.35: --exec-id=756-fa6e193d --log-level-console=
   info --log-level-stderr=off --no-log-timestamp --pg1-path=/var/lib/pgsql/10/
   data --repo1-cipher-pass=<redacted> --repo1-cipher-type=aes-256-cbc --repo1-
   path=/var/lib/pgbackrest --stanza=demo
P00  INFO: check repo1 configuration (primary)
P00  INFO: check repo1 archive for WAL (primary)
P00  INFO: WAL segment 000000010000000000000001 successfully archived to '/var/lib/
   pgbackrest/archive/demo/10-1/0000000100000000/000000010000000000000001-341
   c7f9adadf9a24eb8374143f9d570cc352a274.gz' on repo1
P00  INFO: check command end: completed successfully
```

6.9 Perform a Backup

By default pgBackRest will wait for the next regularly scheduled checkpoint before starting a backup. Depending on the `checkpoint_timeout` and `checkpoint_segments` settings in PostgreSQL it may be quite some time before a checkpoint completes and the backup can begin. Generally, it is best to set `start-fast=y` so that the backup starts immediately. This forces a checkpoint, but since backups are usually run once a day an additional checkpoint should not have a noticeable impact on performance. However, on very busy clusters it may be best to pass `--start-fast` on the command-line as needed.

pg-primary`:/etc/pgbackrest/pgbackrest.conf` — *Configure backup fast start*

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

[global]
repo1-cipher-pass=zWaf6XtpjIVZC5444yXB+cgFDFl7MxGlgkZSaoPvTGirhPygu4jOKOXf9LO4vjfO
repo1-cipher-type=aes-256-cbc
repo1-path=/var/lib/pgbackrest
repo1-retention-full=2
start-fast=y

[global:archive-push]
compress-level=3
```

To perform a backup of the PostgreSQL cluster run pgBackRest with the `backup` command.

pg-primary — *Backup the demo cluster*

```
sudo -u postgres pgbackrest --stanza=demo \
   --log-level-console=info backup

Output:

P00  INFO: backup command begin 2.35: --exec-id=806-61c63eb7 --log-level-console=
   info --log-level-stderr=off --no-log-timestamp --pg1-path=/var/lib/pgsql/10/
   data --repo1-cipher-pass=<redacted> --repo1-cipher-type=aes-256-cbc --repo1-
   path=/var/lib/pgbackrest --repo1-retention-full=2 --stanza=demo --start-fast
P00  WARN: no prior backup exists, incr backup has been changed to full
P00  INFO: execute non-exclusive pg_start_backup(): backup begins after the
   requested immediate checkpoint completes
P00  INFO: backup start archive = 000000010000000000000002, lsn = 0/2000028
     [filtered 2 lines of output]
P00  INFO: check archive for segment(s)
   000000010000000000000002:000000010000000000000002
P00  INFO: new backup label = 20210908-153129F
P00  INFO: full backup size = 22.4MB, file total = 949
P00  INFO: backup command end: completed successfully
P00  INFO: expire command begin 2.35: --exec-id=806-61c63eb7 --log-level-console=
   info --log-level-stderr=off --no-log-timestamp --repo1-cipher-pass=<redacted>
   --repo1-cipher-type=aes-256-cbc --repo1-path=/var/lib/pgbackrest --repo1-
   retention-full=2 --stanza=demo
```

By default pgBackRest will attempt to perform an incremental backup. However, an incremental backup must be based on a full backup and since no full backup existed pgBackRest ran a full backup instead.

The `type` option can be used to specify a full or differential backup.

pg-primary — *Differential backup of the demo cluster*

```
sudo -u postgres pgbackrest --stanza=demo --type=diff \
   --log-level-console=info backup

Output:

     [filtered 6 lines of output]
P00  INFO: check archive for segment(s)
   000000010000000000000003:000000010000000000000003
P00  INFO: new backup label = 20210908-153129F_20210908-153134D
P00  INFO: diff backup size = 8.6KB, file total = 949
P00  INFO: backup command end: completed successfully
P00  INFO: expire command begin 2.35: --exec-id=851-519079c2 --log-level-console=
   info --log-level-stderr=off --no-log-timestamp --repo1-cipher-pass=<redacted>
   --repo1-cipher-type=aes-256-cbc --repo1-path=/var/lib/pgbackrest --repo1-
   retention-full=2 --stanza=demo
```

This time there was no warning because a full backup already existed. While incremental backups can be based on a full *or* differential backup, differential backups must be based on a full backup. A full

backup can be performed by running the `backup` command with `--type=full`.

During an online backup pgBackRest waits for WAL segments that are required for backup consistency to be archived. This wait time is governed by the pgBackRest `archive-timeout` option which defaults to 60 seconds. If archiving an individual segment is known to take longer then this option should be increased.

6.10 Schedule a Backup

Backups can be scheduled with utilities such as cron.

In the following example, two cron jobs are configured to run; full backups are scheduled for 6:30 AM every Sunday with differential backups scheduled for 6:30 AM Monday through Saturday. If this crontab is installed for the first time mid-week, then pgBackRest will run a full backup the first time the differential job is executed, followed the next day by a differential backup.

crontab:

```
#m h  dom mon dow command
            30 06 * *  0    pgbackrest --type=full --stanza=demo backup
            30 06 * *  1-6  pgbackrest --type=diff --stanza=demo backup
```

Once backups are scheduled it's important to configure retention so backups are expired on a regular schedule, see RETENTION.

6.11 Backup Information

Use the `info` command to get information about backups.

pg-primary — *Get info for the demo cluster*

```
sudo -u postgres pgbackrest info

Output:

stanza: demo
   status: ok
   cipher: aes-256-cbc

   db (current)
      wal archive min/max (10): 000000010000000000000001/000000010000000000000003

      full backup: 20210908-153129F
         timestamp start/stop: 2021-09-08 15:31:29 / 2021-09-08 15:31:33
         wal start/stop: 000000010000000000000002 / 000000010000000000000002
         database size: 22.4MB, database backup size: 22.4MB
         repo1: backup set size: 2.7MB, backup size: 2.7MB
```

```
    diff backup: 20210908-153129F_20210908-153134D
       timestamp start/stop: 2021-09-08 15:31:34 / 2021-09-08 15:31:36
       wal start/stop: 000000010000000000000003 / 000000010000000000000003
       database size: 22.4MB, database backup size: 8.8KB
       repo1: backup set size: 2.7MB, backup size: 800B
       backup reference list: 20210908-153129F
```

The `info` command operates on a single stanza or all stanzas. Text output is the default and gives a human-readable summary of backups for the stanza(s) requested. This format is subject to change with any release.

For machine-readable output use `--output=json`. The JSON output contains far more information than the text output and is kept stable unless a bug is found.

Each stanza has a separate section and it is possible to limit output to a single stanza with the `--stanza` option. The stanza '`status`' gives a brief indication of the stanza's health. If this is '`ok`' then pgBackRest is functioning normally. If there are multiple repositories, then a status of '`mixed`' indicates that the stanza is not in a healthy state on one or more of the repositories; in this case the state of the stanza will be detailed per repository. For cases in which an error on a repository occurred that is not one of the known error codes, then an error code of '`other`' will be used and the full error details will be provided. The '`wal archive min/max`' shows the minimum and maximum WAL currently stored in the archive and, in the case of multiple repositories, will be reported across all repositories unless the `--repo` option is set. Note that there may be gaps due to archive retention policies or other reasons.

The '`backup/expire running`' message will appear beside the '`status`' information if one of those commands is currently running on the host.

The backups are displayed oldest to newest. The oldest backup will *always* be a full backup (indicated by an `F` at the end of the label) but the newest backup can be full, differential (ends with `D`), or incremental (ends with `I`).

The '`timestamp start/stop`' defines the time period when the backup ran. The '`timestamp stop`' can be used to determine the backup to use when performing Point-In-Time Recovery. More information about Point-In-Time Recovery can be found in the POINT-IN-TIME RECOVERY section.

The '`wal start/stop`' defines the WAL range that is required to make the database consistent when restoring. The `backup` command will ensure that this WAL range is in the archive before completing.

The '`database size`' is the full uncompressed size of the database while '`database backup size`' is the amount of data in the database to actually back up (these will be the same for full backups).

The '`repo`' indicates in which repository this backup resides. The '`backup set size`' includes all the files from this backup and any referenced backups in the repository that are required to restore the database from this backup while '`backup size`' includes only the files in this backup (these will also be the same for full backups). Repository sizes reflect compressed file sizes if compression is enabled in pgBackRest or the filesystem.

The '`backup reference list`' contains the additional backups that are required to restore this backup.

6.12 Restore a Backup

Backups can protect you from a number of disaster scenarios, the most common of which are hardware failure and data corruption. The easiest way to simulate data corruption is to remove an important PostgreSQL cluster file.

pg-primary — *Stop the demo cluster and delete the* `pg_control` *file*

```
sudo systemctl stop postgresql-10.service
sudo -u postgres rm /var/lib/pgsql/10/data/global/pg_control
```

Starting the cluster without this important file will result in an error.

pg-primary — *Attempt to start the corrupted demo cluster*

```
sudo systemctl start postgresql-10.service
sudo systemctl status postgresql-10.service

Output:

     [filtered 8 lines of output]
Sep 08 15:31:37 pg-primary systemd[1]: Starting PostgreSQL 10 database server...
Sep 08 15:31:37 pg-primary systemd[1]: postgresql-10.service: main process exited,
    code=exited, status=2/INVALIDARGUMENT
Sep 08 15:31:37 pg-primary systemd[1]: Failed to start PostgreSQL 10 database
   server.
Sep 08 15:31:37 pg-primary systemd[1]: Unit postgresql-10.service entered failed
   state.
Sep 08 15:31:37 pg-primary systemd[1]: postgresql-10.service failed.
```

To restore a backup of the PostgreSQL cluster run pgBackRest with the `restore` command. The cluster needs to be stopped (in this case it is already stopped) and all files must be removed from the PostgreSQL data directory.

pg-primary — *Remove old files from demo cluster*

```
sudo -u postgres find /var/lib/pgsql/10/data -mindepth 1 -delete
```

pg-primary — *Restore the demo cluster and start PostgreSQL*

```
sudo -u postgres pgbackrest --stanza=demo restore
sudo systemctl start postgresql-10.service
```

This time the cluster started successfully since the restore replaced the missing `pg_control` file.

More information about the `restore` command can be found in the RESTORE section.

7 Monitoring

Monitoring is an important part of any production system. There are many tools available and pgBackRest can be monitored on any of them with a little work.

pgBackRest can output information about the repository in JSON format which includes a list of all backups for each stanza and WAL archive info.

7.1 In PostgreSQL

The PostgreSQL `COPY` command allows pgBackRest info to be loaded into a table. The following example wraps that logic in a function that can be used to perform real-time queries.

pg-primary — *Load pgBackRest info function for PostgreSQL*

```
sudo -u postgres cat \
   /var/lib/pgsql/pgbackrest/doc/example/pgsql-pgbackrest-info.sql

Output:

-- An example of monitoring pgBackRest from within PostgreSQL
--
-- Use copy to export data from the pgBackRest info command into the jsonb
-- type so it can be queried directly by PostgreSQL.

-- Create monitor schema
create schema monitor;

-- Get pgBackRest info in JSON format
create function monitor.pgbackrest_info()
   returns jsonb AS $$
declare
   data jsonb;
begin
   -- Create a temp table to hold the JSON data
   create temp table temp_pgbackrest_data (data jsonb);

   -- Copy data into the table directly from the pgBackRest info command
   copy temp_pgbackrest_data (data)
      from program
         'pgbackrest --output=json info' (format text);

   select temp_pgbackrest_data.data
     into data
     from temp_pgbackrest_data;

   drop table temp_pgbackrest_data;
```

```
   return data;
end $$ language plpgsql;
sudo -u postgres psql -f \
   /var/lib/pgsql/pgbackrest/doc/example/pgsql-pgbackrest-info.sql
```

Now the `monitor.pgbackrest_info()` function can be used to determine the last successful backup time and archived WAL for a stanza.

pg-primary — *Query last successful backup time and archived WAL*

```
sudo -u postgres cat \
   /var/lib/pgsql/pgbackrest/doc/example/pgsql-pgbackrest-query.sql

Output:

-- Get last successful backup for each stanza
--
-- Requires the monitor.pgbackrest_info function.
with stanza as
(
   select data->'name' as name,
        data->'backup'->(
           jsonb_array_length(data->'backup') - 1) as last_backup,
        data->'archive'->(
           jsonb_array_length(data->'archive') - 1) as current_archive
    from jsonb_array_elements(monitor.pgbackrest_info()) as data
)
select name,
     to_timestamp(
        (last_backup->'timestamp'->>'stop')::numeric) as last_successful_backup,
     current_archive->>'max' as last_archived_wal
  from stanza;
sudo -u postgres psql -f \
   /var/lib/pgsql/pgbackrest/doc/example/pgsql-pgbackrest-query.sql

Output:

 name | last_successful_backup | last_archived_wal
--------+------------------------+--------------------------
 "demo" | 2021-09-08 15:31:36+00 | 000000010000000000000004
(1 row)
```

8 Retention

Generally it is best to retain as many backups as possible to provide a greater window for POINT-IN-TIME RECOVERY, but practical concerns such as disk space must also be considered. Retention options remove older backups once they are no longer needed.

pgBackRest does full backup rotation based on the retention type which can be a count or a time period. When a count is specified, then expiration is not concerned with when the backups were created but with how many must be retained. Differential and Incremental backups are count-based but will always be expired when the backup they depend on is expired. See sections FULL BACKUP RETENTION and DIFFERENTIAL BACKUP RETENTION for details and examples. Archived WAL is retained by default for backups that have not expired, however, although not recommended, this schedule can be modified per repository with the retention-archive options. See section ARCHIVE RETENTION for details and examples.

The `expire` command is run automatically after each successful backup and can also be run by the user. When run by the user, expiration will occur as defined by the retention settings for each configured repository. If the `--repo` option is provided, expiration will occur only on the specified repository. Expiration can also be limited by the user to a specific backup set with the `--set` option and, unless the `--repo` option is specified, all repositories will be searched and any matching the set criteria will be expired. It should be noted that the archive retention schedule will be checked and performed any time the `expire` command is run.

8.1 Full Backup Retention

The `repo1-retention-full-type` determines how the option `repo1-retention-full` is interpreted; either as the count of full backups to be retained or how many days to retain full backups. New backups must be completed before expiration will occur — that means if `repo1-retention-full-type=count` and `repo1-retention-full=2` then there will be three full backups stored before the oldest one is expired, or if `repo1-retention-full-type=time` and `repo1-retention-full=20` then there must be one full backup that is at least 20 days old before expiration can occur.

pg-primary*:/etc/pgbackrest/pgbackrest.conf — Configure* `repo1-retention-full`

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

[global]
repo1-cipher-pass=zWaf6XtpjIVZC5444yXB+cgFDFl7MxGlgkZSaoPvTGirhPygu4jOKOXf9LO4vjfO
repo1-cipher-type=aes-256-cbc
repo1-path=/var/lib/pgbackrest
repo1-retention-full=2
start-fast=y

[global:archive-push]
```

```
compress-level=3
```

Backup `repo1-retention-full=2` but currently there is only one full backup so the next full backup to run will not expire any full backups.

pg-primary — *Perform a full backup*

```
sudo -u postgres pgbackrest --stanza=demo --type=full \
   --log-level-console=detail backup

Output:

     [filtered 957 lines of output]
P00  INFO: backup command end: completed successfully
P00  INFO: expire command begin 2.35: --exec-id=1221-9bc56c5b --log-level-console=
   detail --log-level-stderr=off --no-log-timestamp --repo1-cipher-pass=<redacted>
    --repo1-cipher-type=aes-256-cbc --repo1-path=/var/lib/pgbackrest --repo1-
   retention-full=2 --stanza=demo
P00 DETAIL: repo1: 10-1 archive retention on backup 20210908-153129F, start =
   000000010000000000000002
P00  INFO: repo1: 10-1 remove archive, start = 000000010000000000000001, stop =
   000000010000000000000001
P00  INFO: expire command end: completed successfully
```

Archive *is* expired because WAL segments were generated before the oldest backup. These are not useful for recovery — only WAL segments generated after a backup can be used to recover that backup.

pg-primary — *Perform a full backup*

```
sudo -u postgres pgbackrest --stanza=demo --type=full \
   --log-level-console=info backup

Output:

     [filtered 8 lines of output]
P00  INFO: backup command end: completed successfully
P00  INFO: expire command begin 2.35: --exec-id=1267-32134215 --log-level-console=
   info --log-level-stderr=off --no-log-timestamp --repo1-cipher-pass=<redacted>
   --repo1-cipher-type=aes-256-cbc --repo1-path=/var/lib/pgbackrest --repo1-
   retention-full=2 --stanza=demo
P00  INFO: repo1: expire full backup set 20210908-153129F, 20210908-153129
   F_20210908-153134D
P00  INFO: repo1: remove expired backup 20210908-153129F_20210908-153134D
P00  INFO: repo1: remove expired backup 20210908-153129F
P00  INFO: repo1: 10-1 remove archive, start = 0000000100000000, stop =
   000000020000000000000005
P00  INFO: expire command end: completed successfully
```

The `20210908-153129F` full backup is expired and archive retention is based on the `20210908-153145F` which is now the oldest full backup.

8.2 Differential Backup Retention

Set `repo1-retention-diff` to the number of differential backups required. Differentials only rely on the prior full backup so it is possible to create a "rolling" set of differentials for the last day or more. This allows quick restores to recent points-in-time but reduces overall space consumption.

pg-primary`:/etc/pgbackrest/pgbackrest.conf` — *Configure* `repo1-retention-diff`

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

[global]
repo1-cipher-pass=zWaf6XtpjIVZC5444yXB+cgFDFl7MxGlgkZSaoPvTGirhPygu4jOKOXf9LO4vjfO
repo1-cipher-type=aes-256-cbc
repo1-path=/var/lib/pgbackrest
repo1-retention-diff=1
repo1-retention-full=2
start-fast=y

[global:archive-push]
compress-level=3
```

Backup `repo1-retention-diff=1` so two differentials will need to be performed before one is expired. An incremental backup is added to demonstrate incremental expiration. Incremental backups cannot be expired independently — they are always expired with their related full or differential backup.

pg-primary — *Perform differential and incremental backups*

```
sudo -u postgres pgbackrest --stanza=demo --type=diff backup
sudo -u postgres pgbackrest --stanza=demo --type=incr backup
```

Now performing a differential backup will expire the previous differential and incremental backups leaving only one differential backup.

pg-primary — *Perform a differential backup*

```
sudo -u postgres pgbackrest --stanza=demo --type=diff \
   --log-level-console=info backup

Output:

     [filtered 9 lines of output]
P00  INFO: backup command end: completed successfully
P00  INFO: expire command begin 2.35: --exec-id=1388-09052fb0 --log-level-console=
   info --log-level-stderr=off --no-log-timestamp --repo1-cipher-pass=<redacted>
   --repo1-cipher-type=aes-256-cbc --repo1-path=/var/lib/pgbackrest --repo1-
   retention-diff=1 --repo1-retention-full=2 --stanza=demo
P00  INFO: repo1: expire diff backup set 20210908-153150F_20210908-153156D,
   20210908-153150F_20210908-153159I
P00  INFO: repo1: remove expired backup 20210908-153150F_20210908-153159I
```

```
P00  INFO: repo1: remove expired backup 20210908-153150F_20210908-153156D
P00  INFO: expire command end: completed successfully
```

8.3 Archive Retention

Although pgBackRest automatically removes archived WAL segments when expiring backups (the default expires WAL for full backups based on the `repo1-retention-full` option), it may be useful to expire archive more aggressively to save disk space. Note that full backups are treated as differential backups for the purpose of differential archive retention.

Expiring archive will never remove WAL segments that are required to make a backup consistent. However, since Point-in-Time-Recovery (PITR) only works on a continuous WAL stream, care should be taken when aggressively expiring archive outside of the normal backup expiration process. To determine what will be expired without actually expiring anything, the `dry-run` option can be provided on the command line with the `expire` command.

pg-primary*:/etc/pgbackrest/pgbackrest.conf* — *Configure* `repo1-retention-diff`

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

[global]
repo1-cipher-pass=zWaf6XtpjIVZC5444yXB+cgFDFl7MxGlgkZSaoPvTGirhPygu4jOKOXf9LO4vjfO
repo1-cipher-type=aes-256-cbc
repo1-path=/var/lib/pgbackrest
repo1-retention-diff=2
repo1-retention-full=2
start-fast=y

[global:archive-push]
compress-level=3
```

pg-primary — *Perform differential backup*

```
sudo -u postgres pgbackrest --stanza=demo --type=diff \
   --log-level-console=info backup

Output:

     [filtered 5 lines of output]
P00  INFO: backup stop archive = 000000020000000000000000E, lsn = 0/E0000F8
P00  INFO: check archive for segment(s) 000000020000000000000000E
   :000000020000000000000000E
P00  INFO: new backup label = 20210908-153150F_20210908-153205D
P00  INFO: diff backup size = 10.1KB, file total = 949
P00  INFO: backup command end: completed successfully
     [filtered 2 lines of output]
```

pg-primary — *Expire archive*

```
sudo -u postgres pgbackrest --stanza=demo --log-level-console=detail \
   --repo1-retention-archive-type=diff --repo1-retention-archive=1 expire

Output:

P00  INFO: expire command begin 2.35: --exec-id=1527-74dc2f47 --log-level-console=
   detail --log-level-stderr=off --no-log-timestamp --repo1-cipher-pass=<redacted>
    --repo1-cipher-type=aes-256-cbc --repo1-path=/var/lib/pgbackrest --repo1-
   retention-archive=1 --repo1-retention-archive-type=diff --repo1-retention-diff
   =2 --repo1-retention-full=2 --stanza=demo
P00 DETAIL: repo1: 10-1 archive retention on backup 20210908-153145F, start =
   000000020000000000000006, stop = 000000020000000000000006
P00 DETAIL: repo1: 10-1 archive retention on backup 20210908-153150F, start =
   000000020000000000000007, stop = 000000020000000000000007
P00 DETAIL: repo1: 10-1 archive retention on backup 20210908-153150F_20210908
   -153201D, start = 00000002000000000000000B, stop = 00000002000000000000000B
P00 DETAIL: repo1: 10-1 archive retention on backup 20210908-153150F_20210908
   -153205D, start = 00000002000000000000000E
P00  INFO: repo1: 10-1 remove archive, start = 000000020000000000000008, stop =
   00000002000000000000000A
P00  INFO: repo1: 10-1 remove archive, start = 00000002000000000000000C, stop =
   00000002000000000000000D
P00  INFO: expire command end: completed successfully
```

The `20210908-153150F_20210908-153201D` differential backup has archived WAL segments that must be retained to make the older backups consistent even though they cannot be played any further forward with PITR. WAL segments generated after `20210908-153150F_20210908-153201D` but before `20210908-153150F_20210908-153205D` are removed. WAL segments generated after the new backup `20210908-153150F_20210908-153205D` remain and can be used for PITR.

Since full backups are considered differential backups for the purpose of differential archive retention, if a full backup is now performed with the same settings, only the archive for that full backup is retained for PITR.

9 Restore

The restore command automatically defaults to selecting the latest backup from the first repository where backups exist (see QUICK START - RESTORE A BACKUP). The order in which the repositories are checked is dictated by the `pgbackrest.conf` (e.g. repo1 will be checked before repo2). To select from a specific repository, the `--repo` option can be passed (e.g. `--repo=1`). The `--set` option can be passed if a backup other than the latest is desired.

When PITR of `--type=time` is specified, then the target time must be specified with the `--target` option. If a backup is not specified via the `--set` option, then the configured repositories will be checked, in order, for a backup that contains the requested time. If no backup can be found, the latest backup from the first repository containing backups will be used. For other types of PITR, e.g. `xid`, the `--set` option must be provided if the target is prior to the latest backup. See POINT-IN-TIME RECOVERY for more details and examples.

The following sections introduce additional `restore` command features.

9.1 File Ownership

If a `restore` is run as a non-root user (the typical scenario) then all files restored will belong to the user/group executing pgBackRest. If existing files are not owned by the executing user/group then an error will result if the ownership cannot be updated to the executing user/group. In that case the file ownership will need to be updated by a privileged user before the restore can be retried.

If a `restore` is run as the `root` user then pgBackRest will attempt to recreate the ownership recorded in the manifest when the backup was made. Only user/group **names** are stored in the manifest so the same names must exist on the restore host for this to work. If the user/group name cannot be found locally then the user/group of the PostgreSQL data directory will be used and finally `root` if the data directory user/group cannot be mapped to a name.

9.2 Delta Option

RESTORE A BACKUP in QUICK START required the database cluster directory to be cleaned before the `restore` could be performed. The `delta` option allows pgBackRest to automatically determine which files in the database cluster directory can be preserved and which ones need to be restored from the backup — it also *removes* files not present in the backup manifest so it will dispose of divergent changes. This is accomplished by calculating a SHA-1 cryptographic hash for each file in the database cluster directory. If the `SHA-1` hash does not match the hash stored in the backup then that file will be restored. This operation is very efficient when combined with the `process-max` option. Since the PostgreSQL server is shut down during the restore, a larger number of processes can be used than might be desirable during a backup when the PostgreSQL server is running.

pg-primary — *Stop the demo cluster, perform delta restore*

```
sudo systemctl stop postgresql-10.service
```

```
sudo -u postgres pgbackrest --stanza=demo --delta \
   --log-level-console=detail restore

Output:

     [filtered 2 lines of output]
P00 DETAIL: check '/var/lib/pgsql/10/data' exists
P00 DETAIL: remove 'global/pg_control' so cluster will not start if restore does
   not complete
P00  INFO: remove invalid files/links/paths from '/var/lib/pgsql/10/data'
P00 DETAIL: remove invalid file '/var/lib/pgsql/10/data/backup_label.old'
P00 DETAIL: remove invalid file '/var/lib/pgsql/10/data/base/12953/pg_internal.
   init'
     [filtered 997 lines of output]
```

pg-primary — *Restart PostgreSQL*

```
sudo systemctl start postgresql-10.service
```

9.3 Restore Selected Databases

There may be cases where it is desirable to selectively restore specific databases from a cluster backup. This could be done for performance reasons or to move selected databases to a machine that does not have enough space to restore the entire cluster backup.

To demonstrate this feature two databases are created: test1 and test2.

pg-primary — *Create two test databases*

```
sudo -u postgres psql -c "create database test1;"

Output:

CREATE DATABASE
sudo -u postgres psql -c "create database test2;"

Output:

CREATE DATABASE
```

Each test database will be seeded with tables and data to demonstrate that recovery works with selective restore.

pg-primary — *Create a test table in each database*

```
sudo -u postgres psql -c "create table test1_table (id int); \
   insert into test1_table (id) values (1);" test1
```

```
Output:

INSERT 0 1
sudo -u postgres psql -c "create table test2_table (id int); \
   insert into test2_table (id) values (2);" test2

Output:

INSERT 0 1
```

A fresh backup is run so pgBackRest is aware of the new databases.

pg-primary — *Perform a backup*

```
sudo -u postgres pgbackrest --stanza=demo --type=incr backup
```

One of the main reasons to use selective restore is to save space. The size of the test1 database is shown here so it can be compared with the disk utilization after a selective restore.

pg-primary — *Show space used by test1 database*

```
sudo -u postgres du -sh /var/lib/pgsql/10/data/base/24576

Output:

7.5M  /var/lib/pgsql/10/data/base/24576
```

If the database to restore is not known, use the `info` command `set` option to discover databases that are part of the backup set.

pg-primary — *Show database list for backup*

```
sudo -u postgres pgbackrest --stanza=demo \
   --set=20210908-153150F_20210908-153214I info

Output:

     [filtered 11 lines of output]
         repo1: backup set size: 4.4MB, backup size: 1.8MB
         backup reference list: 20210908-153150F, 20210908-153150F_20210908-153205
            D
         database list: postgres (12953), test1 (24576), test2 (24577)
```

Stop the cluster and restore only the test2 database. Built-in databases (`template0`, `template1`, and `postgres`) are always restored.

> ***WARNING:*** *Recovery may error unless* `--type=immediate` *is specified. This is because after consistency is reached PostgreSQL will flag zeroed pages as errors even for a full-page write. For PostgreSQL ≥* `13` *the* `ignore_invalid_pages` *setting may be used to ignore invalid pages. In this case it is important to check the logs after recovery to ensure that no invalid pages were reported in the selected databases.*

pg-primary — *Restore from last backup including only the test2 database*

```
sudo systemctl stop postgresql-10.service
sudo -u postgres pgbackrest --stanza=demo --delta \
   --db-include=test2 --type=immediate --target-action=promote restore
sudo systemctl start postgresql-10.service
```

Once recovery is complete the test2 database will contain all previously created tables and data.

pg-primary — *Demonstrate that the test2 database was recovered*

```
sudo -u postgres psql -c "select * from test2_table;" test2

Output:

 id
----
  2
(1 row)
```

The test1 database, despite successful recovery, is not accessible. This is because the entire database was restored as sparse, zeroed files. PostgreSQL can successfully apply WAL on the zeroed files but the database as a whole will not be valid because key files contain no data. This is purposeful to prevent the database from being accidentally used when it might contain partial data that was applied during WAL replay.

pg-primary — *Attempting to connect to the test1 database will produce an error*

```
sudo -u postgres psql -c "select * from test1_table;" test1

Output:

psql: FATAL: relation mapping file "base/24576/pg_filenode.map" contains invalid
   data
```

Since the test1 database is restored with sparse, zeroed files it will only require as much space as the amount of WAL that is written during recovery. While the amount of WAL generated during a backup and applied during recovery can be significant it will generally be a small fraction of the total database size, especially for large databases where this feature is most likely to be useful.

It is clear that the test1 database uses far less disk space during the selective restore than it would have if the entire database had been restored.

pg-primary — *Show space used by test1 database after recovery*

```
sudo -u postgres du -sh /var/lib/pgsql/10/data/base/24576

Output:

16K   /var/lib/pgsql/10/data/base/24576
```

At this point the only action that can be taken on the invalid test1 database is `drop database`. pgBackRest does not automatically drop the database since this cannot be done until recovery is complete and the cluster is accessible.

pg-primary — *Drop the test1 database*

```
sudo -u postgres psql -c "drop database test1;"

Output:

DROP DATABASE
```

Now that the invalid test1 database has been dropped only the test2 and built-in databases remain.

pg-primary — *List remaining databases*

```
sudo -u postgres psql -c "select oid, datname from pg_database order by oid;"

Output:

  oid | datname
-------+-----------
    1 | template1
 12952 | template0
 12953 | postgres
 24577 | test2
(4 rows)
```

10 Point-in-Time Recovery

RESTORE A BACKUP in QUICK START performed default recovery, which is to play all the way to the end of the WAL stream. In the case of a hardware failure this is usually the best choice but for data corruption scenarios (whether machine or human in origin) Point-in-Time Recovery (PITR) is often more appropriate.

Point-in-Time Recovery (PITR) allows the WAL to be played from the last backup to a specified lsn, time, transaction id, or recovery point. For common recovery scenarios time-based recovery is arguably the most useful. A typical recovery scenario is to restore a table that was accidentally dropped or data that was accidentally deleted. Recovering a dropped table is more dramatic so that's the example given here but deleted data would be recovered in exactly the same way.

pg-primary — *Backup the demo cluster and create a table with very important data*

```
sudo -u postgres pgbackrest --stanza=demo --type=diff backup
sudo -u postgres psql -c "begin; \
   create table important_table (message text); \
   insert into important_table values ('Important Data'); \
   commit; \
   select * from important_table;"

Output:

   message
----------------
 Important Data
(1 row)
```

It is important to represent the time as reckoned by PostgreSQL and to include timezone offsets. This reduces the possibility of unintended timezone conversions and an unexpected recovery result.

pg-primary — *Get the time from PostgreSQL*

```
sudo -u postgres psql -Atc "select current_timestamp"

Output:

2021-09-08 15:32:29.310964+00
```

Now that the time has been recorded the table is dropped. In practice finding the exact time that the table was dropped is a lot harder than in this example. It may not be possible to find the exact time, but some forensic work should be able to get you close.

pg-primary — *Drop the important table*

```
sudo -u postgres psql -c "begin; \
   drop table important_table; \
   commit; \
   select * from important_table;"
```

```
Output:

ERROR: relation "important_table" does not exist
LINE 1: ...le important_table; commit; select * from important_...
                                                 ^
```

Now the restore can be performed with time-based recovery to bring back the missing table.

pg-primary — *Stop PostgreSQL, restore the demo cluster to* `2021-09-08 15:32:29.310964+00`*, and display* `recovery.conf`

```
sudo systemctl stop postgresql-10.service
sudo -u postgres pgbackrest --stanza=demo --delta \
   --type=time "--target=2021-09-08 15:32:29.310964+00" \
   --target-action=promote restore
sudo -u postgres cat /var/lib/pgsql/10/data/recovery.conf

Output:

# Recovery settings generated by pgBackRest restore on 2021-09-08 15:32:32
restore_command = 'pgbackrest --stanza=demo archive-get %f "%p"'
recovery_target_time = '2021-09-08 15:32:29.310964+00'
recovery_target_action = 'promote'
```

pgBackRest has automatically generated the recovery settings in `recovery.conf` so PostgreSQL can be started immediately. `%f` is how PostgreSQL specifies the WAL segment it needs and `%p` is the location where it should be copied. Once PostgreSQL has finished recovery the table will exist again and can be queried.

pg-primary — *Start PostgreSQL and check that the important table exists*

```
sudo systemctl start postgresql-10.service
sudo -u postgres psql -c "select * from important_table"

Output:

   message
----------------
 Important Data
(1 row)
```

The PostgreSQL log also contains valuable information. It will indicate the time and transaction where the recovery stopped and also give the time of the last transaction to be applied.

pg-primary — *Examine the PostgreSQL log output*

```
sudo -u postgres cat /var/lib/pgsql/10/data/log/postgresql.log

Output:
```

```
LOG: database system was interrupted; last known up at 2021-09-08 15:32:25 UTC
LOG: starting point-in-time recovery to 2021-09-08 15:32:29.310964+00
LOG: restored log file "00000004.history" from archive
LOG: restored log file "000000040000000000000011" from archive
     [filtered 2 lines of output]
LOG: database system is ready to accept read only connections
LOG: restored log file "000000040000000000000012" from archive
LOG: recovery stopping before commit of transaction 564, time 2021-09-08
   15:32:30.88732+00
LOG: redo done at 0/12021520
LOG: last completed transaction was at log time 2021-09-08 15:32:27.752837+00
LOG: selected new timeline ID: 5
LOG: archive recovery complete
     [filtered 2 lines of output]
```

This example was rigged to give the correct result. If a backup after the required time is chosen then PostgreSQL will not be able to recover the lost table. PostgreSQL can only play forward, not backward. To demonstrate this the important table must be dropped (again).

pg-primary — *Drop the important table (again)*

```
sudo -u postgres psql -c "begin; \
   drop table important_table; \
   commit; \
   select * from important_table;"

Output:

ERROR: relation "important_table" does not exist
LINE 1: ...le important_table; commit; select * from important_...
                                                ^
```

Now take a new backup and attempt recovery from the new backup by specifying the `--set` option. The `info` command can be used to find the new backup label.

pg-primary — *Perform a backup and get backup info*

```
sudo -u postgres pgbackrest --stanza=demo --type=incr backup
sudo -u postgres pgbackrest info

Output:

stanza: demo
   status: ok
   cipher: aes-256-cbc

   db (current)
      wal archive min/max (10): 000000020000000000000006/000000050000000000000013
```

```
      full backup: 20210908-153145F
         timestamp start/stop: 2021-09-08 15:31:45 / 2021-09-08 15:31:49
         wal start/stop: 000000020000000000000006 / 000000020000000000000006
         database size: 22.4MB, database backup size: 22.4MB
         repo1: backup set size: 2.7MB, backup size: 2.7MB

      full backup: 20210908-153150F
         timestamp start/stop: 2021-09-08 15:31:50 / 2021-09-08 15:31:54
         wal start/stop: 000000020000000000000007 / 000000020000000000000007
         database size: 22.4MB, database backup size: 22.4MB
         repo1: backup set size: 2.7MB, backup size: 2.7MB

      diff backup: 20210908-153150F_20210908-153205D
         timestamp start/stop: 2021-09-08 15:32:05 / 2021-09-08 15:32:07
         wal start/stop: 00000002000000000000000E / 00000002000000000000000E
         database size: 22.4MB, database backup size: 10.4KB
         repo1: backup set size: 2.7MB, backup size: 1KB
         backup reference list: 20210908-153150F

      incr backup: 20210908-153150F_20210908-153214I
         timestamp start/stop: 2021-09-08 15:32:14 / 2021-09-08 15:32:17
         wal start/stop: 000000030000000000000010 / 000000030000000000000010
         database size: 37.0MB, database backup size: 15MB
         repo1: backup set size: 4.4MB, backup size: 1.8MB
         backup reference list: 20210908-153150F, 20210908-153150F_20210908-153205
            D

      diff backup: 20210908-153150F_20210908-153225D
         timestamp start/stop: 2021-09-08 15:32:25 / 2021-09-08 15:32:27
         wal start/stop: 000000040000000000000011 / 000000040000000000000011
         database size: 29.7MB, database backup size: 7.8MB
         repo1: backup set size: 3.5MB, backup size: 947.1KB
         backup reference list: 20210908-153150F

      incr backup: 20210908-153150F_20210908-153236I
         timestamp start/stop: 2021-09-08 15:32:36 / 2021-09-08 15:32:39
         wal start/stop: 000000050000000000000013 / 000000050000000000000013
         database size: 29.7MB, database backup size: 2MB
         repo1: backup set size: 3.5MB, backup size: 218.6KB
         backup reference list: 20210908-153150F, 20210908-153150F_20210908-153225
            D
```

pg-primary — *Attempt recovery from the specified backup*

```
sudo systemctl stop postgresql-10.service
sudo -u postgres pgbackrest --stanza=demo --delta \
   --set=20210908-153150F_20210908-153236I \
   --type=time "--target=2021-09-08 15:32:29.310964+00" --target-action=promote
      restore
```

```
sudo systemctl start postgresql-10.service
sudo -u postgres psql -c "select * from important_table"

Output:

ERROR: relation "important_table" does not exist
LINE 1: select * from important_table
                ^
```

Looking at the log output it's not obvious that recovery failed to restore the table. The key is to look for the presence of the "recovery stopping before..." and "last completed transaction..." log messages. If they are not present then the recovery to the specified point-in-time was not successful.

pg-primary — *Examine the PostgreSQL log output to discover the recovery was not successful*

```
sudo -u postgres cat /var/lib/pgsql/10/data/log/postgresql.log

Output:

LOG: database system was interrupted; last known up at 2021-09-08 15:32:37 UTC
LOG: starting point-in-time recovery to 2021-09-08 15:32:29.310964+00
LOG: restored log file "00000005.history" from archive
LOG: restored log file "000000050000000000000013" from archive
LOG: redo starts at 0/13000028
LOG: consistent recovery state reached at 0/130000F8
LOG: database system is ready to accept read only connections
LOG: redo done at 0/130000F8
     [filtered 7 lines of output]
```

The default behavior for time-based restore, if the `--set` option is not specified, is to attempt to discover an earlier backup to play forward from. If a backup set cannot be found, then restore will default to the latest backup which, as shown earlier, may not give the desired result.

pg-primary — *Stop PostgreSQL, restore from auto-selected backup, and start PostgreSQL*

```
sudo systemctl stop postgresql-10.service
sudo -u postgres pgbackrest --stanza=demo --delta \
   --type=time "--target=2021-09-08 15:32:29.310964+00" \
   --target-action=promote restore
sudo systemctl start postgresql-10.service
sudo -u postgres psql -c "select * from important_table"

Output:

   message
----------------
 Important Data
(1 row)
```

Now the log output will contain the expected “recovery stopping before...” and “last completed transaction...” messages showing that the recovery was successful.

pg-primary — *Examine the PostgreSQL log output for log messages indicating success*

```
sudo -u postgres cat /var/lib/pgsql/10/data/log/postgresql.log

Output:

LOG: database system was interrupted; last known up at 2021-09-08 15:32:25 UTC
LOG: starting point-in-time recovery to 2021-09-08 15:32:29.310964+00
LOG: restored log file "00000004.history" from archive
LOG: restored log file "000000040000000000000011" from archive
     [filtered 2 lines of output]
LOG: database system is ready to accept read only connections
LOG: restored log file "000000040000000000000012" from archive
LOG: recovery stopping before commit of transaction 564, time 2021-09-08
   15:32:30.88732+00
LOG: redo done at 0/12021520
LOG: last completed transaction was at log time 2021-09-08 15:32:27.752837+00
LOG: restored log file "00000005.history" from archive
LOG: restored log file "00000006.history" from archive
     [filtered 4 lines of output]
```

11 Multiple Repositories

Multiple repositories may be configured as demonstrated in S3 SUPPORT. A potential benefit is the ability to have a local repository for fast restores and a remote repository for redundancy.

Some commands, e.g. `stanza-create`/`stanza-update`, will automatically work with all configured repositories while others, e.g. STANZA-DELETE, will require a repository to be specified using the `repo` option. See the COMMAND REFERENCE for details on which commands require the repository to be specified.

Note that the `repo` option is not required when only `repo1` is configured in order to maintain backward compatibility. However, the `repo` option *is* required when a single repo is configured as, e.g. `repo2`. This is to prevent command breakage if a new repository is added later.

The `archive-push` command will always push WAL to the archive in all configured repositories but backups will need to be scheduled individually for each repository. In many cases this is desirable since backup types and retention will vary by repository. Likewise, restores must specify a repository. It is generally better to specify a repository for restores that has low latency/cost even if that means more recovery time. Only restore testing can determine which repository will be most efficient.

12 Azure-Compatible Object Store Support

pgBackRest supports locating repositories in `Azure-compatible` object stores. The container used to store the repository must be created in advance — pgBackRest will not do it automatically. The repository can be located in the container root (`/`) but it's usually best to place it in a subpath so object store logs or other data can also be stored in the container without conflicts.

pg-primary`:/etc/pgbackrest/pgbackrest.conf` — *Configure* `Azure`

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

[global]
process-max=4
repo1-cipher-pass=zWaf6XtpjIVZC5444yXB+cgFDFl7MxGlgkZSaoPvTGirhPygu4jOKOXf9LO4vjfO
repo1-cipher-type=aes-256-cbc
repo1-path=/var/lib/pgbackrest
repo1-retention-diff=2
repo1-retention-full=2
repo2-azure-account=pgbackrest
repo2-azure-container=demo-container
repo2-azure-key=YXpLZXk=
repo2-path=/demo-repo
repo2-retention-full=4
repo2-storage-host=blob.core.windows.net
repo2-type=azure
start-fast=y

[global:archive-push]
compress-level=3
```

Shared access signatures may be used by setting the `repo2-azure-key-type` option to `sas` and the `repo2-azure-key` option to the shared access signature token.

Commands are run exactly as if the repository were stored on a local disk.

pg-primary — *Create the stanza*

```
sudo -u postgres pgbackrest --stanza=demo --log-level-console=info stanza-create

Output:

     [filtered 2 lines of output]
P00  INFO: stanza 'demo' already exists on repo1 and is valid
P00  INFO: stanza-create for stanza 'demo' on repo2
P00  INFO: stanza-create command end: completed successfully
```

File creation time in object stores is relatively slow so commands benefit by increasing `process-max` to parallelize file creation.

pg-primary — *Backup the demo cluster*

```
sudo -u postgres pgbackrest --stanza=demo --repo=2 \
   --log-level-console=info backup

Output:

P00  INFO: backup command begin 2.35: --exec-id=2813-90d89096 --log-level-console=
   info --log-level-stderr=off --no-log-timestamp --pg1-path=/var/lib/pgsql/10/
   data --process-max=4 --repo=2 --repo2-azure-account=<redacted> --repo2-azure-
   container=demo-container --repo2-azure-key=<redacted> --repo1-cipher-pass=<
   redacted> --repo1-cipher-type=aes-256-cbc --repo1-path=/var/lib/pgbackrest --
   repo2-path=/demo-repo --repo1-retention-diff=2 --repo1-retention-full=2 --repo2
   -retention-full=4 --repo2-storage-host=blob.core.windows.net --repo2-type=azure
    --stanza=demo --start-fast
P00  WARN: no prior backup exists, incr backup has been changed to full
P00  INFO: execute non-exclusive pg_start_backup(): backup begins after the
   requested immediate checkpoint completes
P00  INFO: backup start archive = 000000070000000000000013, lsn = 0/13000028
     [filtered 2 lines of output]
P00  INFO: check archive for segment(s)
   000000070000000000000013:000000070000000000000013
P00  INFO: new backup label = 20210908-153252F
P00  INFO: full backup size = 29.7MB, file total = 1246
P00  INFO: backup command end: completed successfully
P00  INFO: expire command begin 2.35: --exec-id=2813-90d89096 --log-level-console=
   info --log-level-stderr=off --no-log-timestamp --repo=2 --repo2-azure-account=<
   redacted> --repo2-azure-container=demo-container --repo2-azure-key=<redacted>
   --repo1-cipher-pass=<redacted> --repo1-cipher-type=aes-256-cbc --repo1-path=/
   var/lib/pgbackrest --repo2-path=/demo-repo --repo1-retention-diff=2 --repo1-
   retention-full=2 --repo2-retention-full=4 --repo2-storage-host=blob.core.
   windows.net --repo2-type=azure --stanza=demo
```

13 S3-Compatible Object Store Support

pgBackRest supports locating repositories in `S3-compatible` object stores. The bucket used to store the repository must be created in advance — pgBackRest will not do it automatically. The repository can be located in the bucket root (`/`) but it's usually best to place it in a subpath so object store logs or other data can also be stored in the bucket without conflicts.

pg-primary`:/etc/pgbackrest/pgbackrest.conf` — *Configure* `S3`

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

[global]
process-max=4
repo1-cipher-pass=zWaf6XtpjIVZC5444yXB+cgFDFl7MxGlgkZSaoPvTGirhPygu4jOKOXf9LO4vjfO
repo1-cipher-type=aes-256-cbc
repo1-path=/var/lib/pgbackrest
repo1-retention-diff=2
repo1-retention-full=2
repo2-azure-account=pgbackrest
repo2-azure-container=demo-container
repo2-azure-key=YXpLZXk=
repo2-path=/demo-repo
repo2-retention-full=4
repo2-storage-host=blob.core.windows.net
repo2-type=azure
repo3-path=/demo-repo
repo3-retention-full=4
repo3-s3-bucket=demo-bucket
repo3-s3-endpoint=s3.us-east-1.amazonaws.com
repo3-s3-key=accessKey1
repo3-s3-key-secret=verySecretKey1
repo3-s3-region=us-east-1
repo3-type=s3
start-fast=y

[global:archive-push]
compress-level=3
```

> ***NOTE:*** *The region and endpoint will need to be configured to where the bucket is located. The values given here are for the* `us-east-1` *region.*

A role should be created to run pgBackRest and the bucket permissions should be set as restrictively as possible. If the role is associated with an instance in `AWS` then pgBackRest will automatically retrieve temporary credentials when `repo3-s3-key-type=auto`, which means that keys do not need to be explicitly set in `/etc/pgbackrest/pgbackrest.conf`.

This sample `Amazon S3` policy will restrict all reads and writes to the bucket and repository path.

Sample Amazon S3 Policy:

```
{
            "Version": "2012-10-17",
            "Statement": [
               {
                  "Effect": "Allow",
                  "Action": [
                     "s3:ListBucket"
                  ],
                  "Resource": [
                     "arn:aws:s3:::{[s3-bucket]}"
                  ],
                  "Condition": {
                     "StringEquals": {
                        "s3:prefix": [
                           "",
                           "{[s3-repo]}"
                        ],
                        "s3:delimiter": [
                           "/"
                        ]
                     }
                  }
               },
               {
                  "Effect": "Allow",
                  "Action": [
                     "s3:ListBucket"
                  ],
                  "Resource": [
                     "arn:aws:s3:::{[s3-bucket]}"
                  ],
                  "Condition": {
                     "StringLike": {
                        "s3:prefix": [
                           "{[s3-repo]}/*"
                        ]
                     }
                  }
               },
               {
                  "Effect": "Allow",
                  "Action": [
                     "s3:PutObject",
                     "s3:GetObject",
                     "s3:DeleteObject"
                  ],
                  "Resource": [
```

```
                    "arn:aws:s3:::{[s3-bucket]}/{[s3-repo]}/*"
                ]
            }
        ]
    }
```

Commands are run exactly as if the repository were stored on a local disk.

pg-primary — *Create the stanza*

```
sudo -u postgres pgbackrest --stanza=demo --log-level-console=info stanza-create

Output:

     [filtered 4 lines of output]
P00  INFO: stanza 'demo' already exists on repo2 and is valid
P00  INFO: stanza-create for stanza 'demo' on repo3
P00  INFO: stanza-create command end: completed successfully
```

File creation time in object stores is relatively slow so commands benefit by increasing `process-max` to parallelize file creation.

pg-primary — *Backup the demo cluster*

```
sudo -u postgres pgbackrest --stanza=demo --repo=3 \
   --log-level-console=info backup

Output:

P00  INFO: backup command begin 2.35: --exec-id=2926-aac2ae03 --log-level-console=
   info --log-level-stderr=off --no-log-timestamp --pg1-path=/var/lib/pgsql/10/
   data --process-max=4 --repo=3 --repo2-azure-account=<redacted> --repo2-azure-
   container=demo-container --repo2-azure-key=<redacted> --repo1-cipher-pass=<
   redacted> --repo1-cipher-type=aes-256-cbc --repo1-path=/var/lib/pgbackrest --
   repo2-path=/demo-repo --repo3-path=/demo-repo --repo1-retention-diff=2 --repo1-
   retention-full=2 --repo2-retention-full=4 --repo3-retention-full=4 --repo3-s3-
   bucket=demo-bucket --repo3-s3-endpoint=s3.us-east-1.amazonaws.com --repo3-s3-
   key=<redacted> --repo3-s3-key-secret=<redacted> --repo3-s3-region=us-east-1 --
   repo2-storage-host=blob.core.windows.net --repo2-type=azure --repo3-type=s3 --
   stanza=demo --start-fast
P00  WARN: no prior backup exists, incr backup has been changed to full
P00  INFO: execute non-exclusive pg_start_backup(): backup begins after the
   requested immediate checkpoint completes
P00  INFO: backup start archive = 000000070000000000000015, lsn = 0/15000028
     [filtered 2 lines of output]
P00  INFO: check archive for segment(s)
   000000070000000000000015:000000070000000000000015
P00  INFO: new backup label = 20210908-153304F
P00  INFO: full backup size = 29.7MB, file total = 1246
P00  INFO: backup command end: completed successfully
```

```
P00  INFO: expire command begin 2.35: --exec-id=2926-aac2ae03 --log-level-console=
   info --log-level-stderr=off --no-log-timestamp --repo=3 --repo2-azure-account=<
   redacted> --repo2-azure-container=demo-container --repo2-azure-key=<redacted>
   --repo1-cipher-pass=<redacted> --repo1-cipher-type=aes-256-cbc --repo1-path=/
   var/lib/pgbackrest --repo2-path=/demo-repo --repo3-path=/demo-repo --repo1-
   retention-diff=2 --repo1-retention-full=2 --repo2-retention-full=4 --repo3-
   retention-full=4 --repo3-s3-bucket=demo-bucket --repo3-s3-endpoint=s3.us-east
   -1.amazonaws.com --repo3-s3-key=<redacted> --repo3-s3-key-secret=<redacted> --
   repo3-s3-region=us-east-1 --repo2-storage-host=blob.core.windows.net --repo2-
   type=azure --repo3-type=s3 --stanza=demo
```

14 GCS-Compatible Object Store Support

pgBackRest supports locating repositories in `GCS-compatible` object stores. The bucket used to store the repository must be created in advance — pgBackRest will not do it automatically. The repository can be located in the bucket root (`/`) but it's usually best to place it in a subpath so object store logs or other data can also be stored in the bucket without conflicts.

pg-primary`:/etc/pgbackrest/pgbackrest.conf` — *Configure* `GCS`

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

[global]
process-max=4
repo1-cipher-pass=zWaf6XtpjIVZC5444yXB+cgFDFl7MxGlgkZSaoPvTGirhPygu4jOKOXf9LO4vjfO
repo1-cipher-type=aes-256-cbc
repo1-path=/var/lib/pgbackrest
repo1-retention-diff=2
repo1-retention-full=2
repo2-azure-account=pgbackrest
repo2-azure-container=demo-container
repo2-azure-key=YXpLZXk=
repo2-path=/demo-repo
repo2-retention-full=4
repo2-storage-host=blob.core.windows.net
repo2-type=azure
repo3-path=/demo-repo
repo3-retention-full=4
repo3-s3-bucket=demo-bucket
repo3-s3-endpoint=s3.us-east-1.amazonaws.com
repo3-s3-key=accessKey1
repo3-s3-key-secret=verySecretKey1
repo3-s3-region=us-east-1
repo3-type=s3
repo4-gcs-bucket=demo-bucket
repo4-gcs-key=/etc/pgbackrest/gcs-key.json
repo4-path=/demo-repo
repo4-type=gcs
start-fast=y

[global:archive-push]
compress-level=3
```

When running in `GCE` set `repo4-gcs-key-type=auto` to automatically authenticate using the instance service account.

Commands are run exactly as if the repository were stored on a local disk.

File creation time in object stores is relatively slow so commands benefit by increasing `process-max`

to parallelize file creation.

15 Delete a Stanza

The `stanza-delete` command removes data in the repository associated with a stanza.

> ***WARNING:*** *Use this command with caution — it will permanently remove all backups and archives from the pgBackRest repository for the specified stanza.*

To delete a stanza:

- Shut down the PostgreSQL cluster associated with the stanza (or use –force to override).
- Run the `stop` command on the host where the `stanza-delete` command will be run.
- Run the `stanza-delete` command.

Once the command successfully completes, it is the responsibility of the user to remove the stanza from all pgBackRest configuration files and/or environment variables.

A stanza may only be deleted from one repository at a time. To delete the stanza from multiple repositories, repeat the `stanza-delete` command for each repository while specifying the `--repo` option.

pg-primary — *Stop PostgreSQL cluster to be removed*

```
sudo systemctl stop postgresql-10.service
```

pg-primary — *Stop pgBackRest for the stanza*

```
sudo -u postgres pgbackrest --stanza=demo --log-level-console=info stop

Output:

P00  INFO: stop command begin 2.35: --exec-id=3001-bc8a079f --log-level-console=
   info --log-level-stderr=off --no-log-timestamp --stanza=demo
P00  INFO: stop command end: completed successfully
```

pg-primary — *Delete the stanza from one repository*

```
sudo -u postgres pgbackrest --stanza=demo --repo=1 \
   --log-level-console=info stanza-delete

Output:

P00  INFO: stanza-delete command begin 2.35: --exec-id=3020-af4237a4 --log-level-
   console=info --log-level-stderr=off --no-log-timestamp --pg1-path=/var/lib/
   pgsql/10/data --repo=1 --repo2-azure-account=<redacted> --repo2-azure-container
   =demo-container --repo2-azure-key=<redacted> --repo1-cipher-pass=<redacted> --
   repo1-cipher-type=aes-256-cbc --repo4-gcs-bucket=demo-bucket --repo4-gcs-key=<
   redacted> --repo1-path=/var/lib/pgbackrest --repo2-path=/demo-repo --repo3-path
   =/demo-repo --repo4-path=/demo-repo --repo3-s3-bucket=demo-bucket --repo3-s3-
   endpoint=s3.us-east-1.amazonaws.com --repo3-s3-key=<redacted> --repo3-s3-key-
   secret=<redacted> --repo3-s3-region=us-east-1 --repo2-storage-host=blob.core.
   windows.net --repo2-type=azure --repo3-type=s3 --repo4-type=gcs --stanza=demo
```

```
P00  INFO: stanza-delete command end: completed successfully
```

16 Dedicated Repository Host

The configuration described in QUICKSTART is suitable for simple installations but for enterprise configurations it is more typical to have a dedicated **repository** host where the backups and WAL archive files are stored. This separates the backups and WAL archive from the database server so **database** host failures have less impact. It is still a good idea to employ traditional backup software to backup the **repository** host.

On PostgreSQL hosts, `pg1-path` is required to be the path of the local PostgreSQL cluster and no `pg1-host` should be configured. When configuring a repository host, the pgbackrest configuration file must have the `pg-host` option configured to connect to the primary and standby (if any) hosts. The repository host has the only pgbackrest configuration that should be aware of more than one PostgreSQL host. Order does not matter, e.g. pg1-path/pg1-host, pg2-path/pg2-host can be primary or standby.

16.1 Installation

A new host named **repository** is created to store the cluster backups.

> ***NOTE:*** *The pgBackRest version installed on the* **repository** *host must exactly match the version installed on the PostgreSQL host.*

The `pgbackrest` user is created to own the pgBackRest repository. Any user can own the repository but it is best not to use `postgres` (if it exists) to avoid confusion.

repository — *Create* `pgbackrest` *user*

```
sudo groupadd pgbackrest
sudo adduser -gpgbackrest -n pgbackrest
```

pgBackRest needs to be installed from a package or installed manually as shown here.

repository — *Install dependencies*

```
sudo yum install postgresql-libs
```

repository — *Copy pgBackRest binary from build host*

```
sudo scp build:/build/pgbackrest-release-2.35/src/pgbackrest /usr/bin
sudo chmod 755 /usr/bin/pgbackrest
```

pgBackRest requires log and configuration directories and a configuration file.

repository — *Create pgBackRest configuration file and directories*

```
sudo mkdir -p -m 770 /var/log/pgbackrest
sudo chown pgbackrest:pgbackrest /var/log/pgbackrest
sudo mkdir -p /etc/pgbackrest
sudo mkdir -p /etc/pgbackrest/conf.d
```

```
sudo touch /etc/pgbackrest/pgbackrest.conf
sudo chmod 640 /etc/pgbackrest/pgbackrest.conf
sudo chown pgbackrest:pgbackrest /etc/pgbackrest/pgbackrest.conf
```

repository — *Create the pgBackRest repository*

```
sudo mkdir -p /var/lib/pgbackrest
sudo chmod 750 /var/lib/pgbackrest
sudo chown pgbackrest:pgbackrest /var/lib/pgbackrest
```

16.2 Setup Passwordless SSH

pgBackRest requires passwordless SSH to enable communication between the hosts.

repository — *Create* **repository** *host key pair*

```
sudo -u pgbackrest mkdir -m 750 /home/pgbackrest/.ssh
sudo -u pgbackrest ssh-keygen -f /home/pgbackrest/.ssh/id_rsa \
   -t rsa -b 4096 -N ""
```

pg-primary — *Create* **pg-primary** *host key pair*

```
sudo -u postgres mkdir -m 750 -p /var/lib/pgsql/.ssh
sudo -u postgres ssh-keygen -f /var/lib/pgsql/.ssh/id_rsa \
   -t rsa -b 4096 -N ""
```

Exchange keys between **repository** and **pg-primary**.

repository — *Copy* **pg-primary** *public key to* **repository**

```
(echo -n 'no-agent-forwarding,no-X11-forwarding,no-port-forwarding,' && \
   echo -n 'command="/usr/bin/pgbackrest ${SSH_ORIGINAL_COMMAND#* }" ' && \
   sudo ssh root@pg-primary cat /var/lib/pgsql/.ssh/id_rsa.pub) | \
   sudo -u pgbackrest tee -a /home/pgbackrest/.ssh/authorized_keys
```

pg-primary — *Copy* **repository** *public key to* **pg-primary**

```
(echo -n 'no-agent-forwarding,no-X11-forwarding,no-port-forwarding,' && \
   echo -n 'command="/usr/bin/pgbackrest ${SSH_ORIGINAL_COMMAND#* }" ' && \
   sudo ssh root@repository cat /home/pgbackrest/.ssh/id_rsa.pub) | \
   sudo -u postgres tee -a /var/lib/pgsql/.ssh/authorized_keys
```

Test that connections can be made from **repository** to **pg-primary** and vice versa.

repository — *Test connection from* **repository** *to* **pg-primary**

```
sudo -u pgbackrest ssh postgres@pg-primary
```

pg-primary — *Test connection from* **pg-primary** *to* **repository**

```
sudo -u postgres ssh pgbackrest@repository
```

> ***NOTE:*** *ssh has been configured to only allow pgBackRest to be run via passwordless ssh. This enhances security in the event that one of the service accounts is hijacked.*

16.3 Configuration

The **repository** host must be configured with the **pg-primary** host/user and database path. The primary will be configured as `pg1` to allow a standby to be added later.

repository*:/etc/pgbackrest/pgbackrest.conf* — *Configure* `pg1-host/pg1-host-user` *and* `pg1-path`

```
[demo]
pg1-host=pg-primary
pg1-path=/var/lib/pgsql/10/data

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

The database host must be configured with the repository host/user. The default for the `repo1-host-user` option is `pgbackrest`. If the `postgres` user does restores on the repository host it is best not to also allow the `postgres` user to perform backups. However, the `postgres` user can read the repository directly if it is in the same group as the `pgbackrest` user.

pg-primary*:/etc/pgbackrest/pgbackrest.conf* — *Configure* `repo1-host/repo1-host-user`

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

[global]
log-level-file=detail
repo1-host=repository
```

Commands are run the same as on a single host configuration except that some commands such as `backup` and `expire` are run from the **repository** host instead of the **database** host.

Create the stanza in the new repository.

repository — *Create the stanza*

```
sudo -u pgbackrest pgbackrest --stanza=demo stanza-create
```

Check that the configuration is correct on both the **database** and **repository** hosts. More information about the `check` command can be found in CHECK THE CONFIGURATION.

pg-primary — *Check the configuration*

```
sudo -u postgres pgbackrest --stanza=demo check
```

repository — *Check the configuration*

```
sudo -u pgbackrest pgbackrest --stanza=demo check
```

16.4 Perform a Backup

To perform a backup of the PostgreSQL cluster run pgBackRest with the `backup` command on the **repository** host.

repository — *Backup the demo cluster*

```
sudo -u pgbackrest pgbackrest --stanza=demo backup

Output:

P00  WARN: no prior backup exists, incr backup has been changed to full
```

Since a new repository was created on the **repository** host the warning about the incremental backup changing to a full backup was emitted.

16.5 Restore a Backup

To perform a restore of the PostgreSQL cluster run pgBackRest with the `restore` command on the **database** host.

pg-primary — *Stop the demo cluster, restore, and restart PostgreSQL*

```
sudo systemctl stop postgresql-10.service
sudo -u postgres pgbackrest --stanza=demo --delta restore
sudo systemctl start postgresql-10.service
```

17 Parallel Backup / Restore

pgBackRest offers parallel processing to improve performance of compression and transfer. The number of processes to be used for this feature is set using the `--process-max` option.

It is usually best not to use more than 25% of available CPUs for the `backup` command. Backups don't have to run that fast as long as they are performed regularly and the backup process should not impact database performance, if at all possible.

The restore command can and should use all available CPUs because during a restore the PostgreSQL cluster is shut down and there is generally no other important work being done on the host. If the host contains multiple clusters then that should be considered when setting restore parallelism.

repository — *Perform a backup with single process*

```
sudo -u pgbackrest pgbackrest --stanza=demo --type=full backup
```

repository`:/etc/pgbackrest/pgbackrest.conf` — *Configure pgBackRest to use multiple* `backup` *processes*

```
[demo]
pg1-host=pg-primary
pg1-path=/var/lib/pgsql/10/data

[global]
process-max=3
repo1-path=/var/lib/pgbackrest
repo1-retention-full=2
start-fast=y
```

repository — *Perform a backup with multiple processes*

```
sudo -u pgbackrest pgbackrest --stanza=demo --type=full backup
```

repository — *Get backup info for the demo cluster*

```
sudo -u pgbackrest pgbackrest info

Output:

stanza: demo
   status: ok
   cipher: none

   db (current)
      wal archive min/max (10): 000000080000000000000001A/000000080000000000000001B

      full backup: 20210908-153345F
         timestamp start/stop: 2021-09-08 15:33:45 / 2021-09-08 15:33:50
         wal start/stop: 000000080000000000000001A / 000000080000000000000001A
```

```
        database size: 29.7MB, database backup size: 29.7MB
        repo1: backup set size: 3.5MB, backup size: 3.5MB

    full backup: 20210908-153353F
        timestamp start/stop: 2021-09-08 15:33:53 / 2021-09-08 15:33:59
        wal start/stop: 000000080000000000000001B / 000000080000000000000001B
        database size: 29.7MB, database backup size: 29.7MB
        repo1: backup set size: 3.5MB, backup size: 3.5MB
```

The performance of the last backup should be improved by using multiple processes. For very small backups the difference may not be very apparent, but as the size of the database increases so will time savings.

18 Starting and Stopping

Sometimes it is useful to prevent pgBackRest from running on a system. For example, when failing over from a primary to a standby it's best to prevent pgBackRest from running on the old primary in case PostgreSQL gets restarted or can't be completely killed. This will also prevent pgBackRest from running on `cron`.

pg-primary — *Stop the pgBackRest services*

```
sudo -u postgres pgbackrest stop
```

New pgBackRest processes will no longer run.

repository — *Attempt a backup*

```
sudo -u pgbackrest pgbackrest --stanza=demo backup

Output:

P00  WARN: unable to check pg-1: [StopError] raised from remote-0 protocol on 'pg-
   primary': stop file exists for all stanzas
P00 ERROR: [056]: unable to find primary cluster - cannot proceed
P00  WARN: remote-0 process on 'pg-primary' terminated unexpectedly [0]
```

Specify the `--force` option to terminate any pgBackRest process that are currently running. If pgBackRest is already stopped then stopping again will generate a warning.

pg-primary — *Stop the pgBackRest services again*

```
sudo -u postgres pgbackrest stop

Output:

P00  WARN: stop file already exists for all stanzas
```

Start pgBackRest processes again with the `start` command.

pg-primary — *Start the pgBackRest services*

```
sudo -u postgres pgbackrest start
```

It is also possible to stop pgBackRest for a single stanza.

pg-primary — *Stop pgBackRest services for the* `demo` *stanza*

```
sudo -u postgres pgbackrest --stanza=demo stop
```

New pgBackRest processes for the specified stanza will no longer run.

repository — *Attempt a backup*

```
sudo -u pgbackrest pgbackrest --stanza=demo backup
```

```
Output:

P00  WARN: unable to check pg-1: [StopError] raised from remote-0 protocol on 'pg-
   primary': stop file exists for stanza demo
P00 ERROR: [056]: unable to find primary cluster - cannot proceed
P00  WARN: remote-0 process on 'pg-primary' terminated unexpectedly [0]
```

The stanza must also be specified when starting the pgBackRest processes for a single stanza.

pg-primary — *Start the pgBackRest services for the* `demo` *stanza*

```
sudo -u postgres pgbackrest --stanza=demo start
```

19 Replication

Replication allows multiple copies of a PostgreSQL cluster (called standbys) to be created from a single primary. The standbys are useful for balancing reads and to provide redundancy in case the primary host fails.

19.1 Installation

A new host named **pg-standby** is created to run the standby.

pgBackRest needs to be installed from a package or installed manually as shown here.

pg-standby — *Install dependencies*

```
sudo yum install postgresql-libs
```

pg-standby — *Copy pgBackRest binary from build host*

```
sudo scp build:/build/pgbackrest-release-2.35/src/pgbackrest /usr/bin
sudo chmod 755 /usr/bin/pgbackrest
```

pgBackRest requires log and configuration directories and a configuration file.

pg-standby — *Create pgBackRest configuration file and directories*

```
sudo mkdir -p -m 770 /var/log/pgbackrest
sudo chown postgres:postgres /var/log/pgbackrest
sudo mkdir -p /etc/pgbackrest
sudo mkdir -p /etc/pgbackrest/conf.d
sudo touch /etc/pgbackrest/pgbackrest.conf
sudo chmod 640 /etc/pgbackrest/pgbackrest.conf
sudo chown postgres:postgres /etc/pgbackrest/pgbackrest.conf
```

19.2 Setup Passwordless SSH

pgBackRest requires passwordless SSH to enable communication between the hosts.

pg-standby — *Create* **pg-standby** *host key pair*

```
sudo -u postgres mkdir -m 750 -p /var/lib/pgsql/.ssh
sudo -u postgres ssh-keygen -f /var/lib/pgsql/.ssh/id_rsa \
   -t rsa -b 4096 -N ""
```

Exchange keys between **repository** and **pg-standby**.

repository — *Copy* **pg-standby** *public key to* **repository**

```
(echo -n 'no-agent-forwarding,no-X11-forwarding,no-port-forwarding,' && \
   echo -n 'command="/usr/bin/pgbackrest ${SSH_ORIGINAL_COMMAND#* }" ' && \
   sudo ssh root@pg-standby cat /var/lib/pgsql/.ssh/id_rsa.pub) | \
   sudo -u pgbackrest tee -a /home/pgbackrest/.ssh/authorized_keys
```

pg-standby — *Copy* **repository** *public key to* **pg-standby**

```
(echo -n 'no-agent-forwarding,no-X11-forwarding,no-port-forwarding,' && \
   echo -n 'command="/usr/bin/pgbackrest ${SSH_ORIGINAL_COMMAND#* }" ' && \
   sudo ssh root@repository cat /home/pgbackrest/.ssh/id_rsa.pub) | \
   sudo -u postgres tee -a /var/lib/pgsql/.ssh/authorized_keys
```

Test that connections can be made from **repository** to **pg-standby** and vice versa.

repository — *Test connection from* **repository** *to* **pg-standby**

```
sudo -u pgbackrest ssh postgres@pg-standby
```

pg-standby — *Test connection from* **pg-standby** *to* **repository**

```
sudo -u postgres ssh pgbackrest@repository
```

19.3 Hot Standby

A hot standby performs replication using the WAL archive and allows read-only queries.

pgBackRest configuration is very similar to **pg-primary** except that the `standby` recovery type will be used to keep the cluster in recovery mode when the end of the WAL stream has been reached.

pg-standby`:/etc/pgbackrest/pgbackrest.conf` — *Configure pgBackRest on the standby*

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

[global]
log-level-file=detail
repo1-host=repository
```

Create the path where PostgreSQL will be restored.

pg-standby — *Create PostgreSQL path*

```
sudo -u postgres mkdir -p -m 700 /var/lib/pgsql/10/data
```

Now the standby can be created with the `restore` command.

IMPORTANT: *If the cluster is intended to be promoted without becoming the new primary (e.g. for reporting or testing), use* `--archive-mode=off` *or set* `archive_mode=off` *in* `postgresql.conf` *to disable archiving. If archiving is not disabled then the repository may be polluted with WAL that can make restores more difficult.*

pg-standby — *Restore the demo standby cluster*

```
sudo -u postgres pgbackrest --stanza=demo --type=standby restore
sudo -u postgres cat /var/lib/pgsql/10/data/recovery.conf

Output:

# Recovery settings generated by pgBackRest restore on 2021-09-08 15:34:15
restore_command = 'pgbackrest --stanza=demo archive-get %f "%p"'
standby_mode = 'on'
```

The `hot_standby` setting must be enabled before starting PostgreSQL to allow read-only connections on **pg-standby**. Otherwise, connection attempts will be refused. The rest of the configuration is in case the standby is promoted to a primary.

pg-standby`:/var/lib/pgsql/10/data/postgresql.conf` — *Configure PostgreSQL*

```
archive_command = 'pgbackrest --stanza=demo archive-push %p'
archive_mode = on
hot_standby = on
log_filename = 'postgresql.log'
log_line_prefix = ''
max_wal_senders = 3
wal_level = replica
```

pg-standby — *Start PostgreSQL*

```
sudo systemctl start postgresql-10.service
```

The PostgreSQL log gives valuable information about the recovery. Note especially that the cluster has entered standby mode and is ready to accept read-only connections.

pg-standby — *Examine the PostgreSQL log output for log messages indicating success*

```
sudo -u postgres cat /var/lib/pgsql/10/data/log/postgresql.log

Output:

LOG: database system was interrupted; last known up at 2021-09-08 15:33:53 UTC
LOG: entering standby mode
LOG: restored log file "00000008.history" from archive
LOG: restored log file "000000080000000000000001B" from archive
LOG: redo starts at 0/1B000028
LOG: consistent recovery state reached at 0/1B000130
LOG: database system is ready to accept read only connections
```

An easy way to test that replication is properly configured is to create a table on **pg-primary**.

pg-primary — *Create a new table on the primary*

```
sudo -u postgres psql -c " \
   begin; \
   create table replicated_table (message text); \
   insert into replicated_table values ('Important Data'); \
   commit; \
   select * from replicated_table";

Output:

   message
----------------
 Important Data
(1 row)
```

And then query the same table on **pg-standby**.

pg-standby — *Query new table on the standby*

```
sudo -u postgres psql -c "select * from replicated_table;"

Output:

ERROR: relation "replicated_table" does not exist
LINE 1: select * from replicated_table;
                      ^
```

So, what went wrong? Since PostgreSQL is pulling WAL segments from the archive to perform replication, changes won't be seen on the standby until the WAL segment that contains those changes is pushed from **pg-primary**.

This can be done manually by calling `pg_switch_wal()` which pushes the current WAL segment to the archive (a new WAL segment is created to contain further changes).

pg-primary — *Call* `pg_switch_wal()`

```
sudo -u postgres psql -c "select *, current_timestamp from pg_switch_wal()";

Output:

 pg_switch_wal |  current_timestamp
---------------+-------------------------------
 0/1C02B690 | 2021-09-08 15:34:21.865485+00
(1 row)
```

Now after a short delay the table will appear on **pg-standby**.

pg-standby — *Now the new table exists on the standby (may require a few retries)*

```
sudo -u postgres psql -c " \
   select *, current_timestamp from replicated_table"

Output:

   message  |     current_timestamp
----------------+-------------------------------
 Important Data | 2021-09-08 15:34:24.596855+00
(1 row)
```

Check the standby configuration for access to the repository.

pg-standby — *Check the configuration*

```
sudo -u postgres pgbackrest --stanza=demo --log-level-console=info check

Output:

P00  INFO: check command begin 2.35: --exec-id=780-5ec86433 --log-level-console=
   info --log-level-file=detail --log-level-stderr=off --no-log-timestamp --pg1-
   path=/var/lib/pgsql/10/data --repo1-host=repository --stanza=demo
P00  INFO: check repo1 (standby)
P00  INFO: switch wal not performed because this is a standby
P00  INFO: check command end: completed successfully
```

19.4 Streaming Replication

Instead of relying solely on the WAL archive, streaming replication makes a direct connection to the primary and applies changes as soon as they are made on the primary. This results in much less lag between the primary and standby.

Streaming replication requires a user with the replication privilege.

pg-primary — *Create replication user*

```
sudo -u postgres psql -c " \
   create user replicator password 'jw8s0F4' replication";

Output:

CREATE ROLE
```

The `pg_hba.conf` file must be updated to allow the standby to connect as the replication user. Be sure to replace the IP address below with the actual IP address of your **pg-primary**. A reload will be required after modifying the `pg_hba.conf` file.

pg-primary — *Create* `pg_hba.conf` *entry for replication user*

```
sudo -u postgres sh -c 'echo \
   "host  replication replicator  172.17.0.7/32     md5" \
   >> /var/lib/pgsql/10/data/pg_hba.conf'
sudo systemctl reload postgresql-10.service
```

The standby needs to know how to contact the primary so the `primary_conninfo` setting will be configured in pgBackRest.

pg-standby`:/etc/pgbackrest/pgbackrest.conf` — *Set* `primary_conninfo`

```
[demo]
pg1-path=/var/lib/pgsql/10/data
recovery-option=primary_conninfo=host=172.17.0.5 port=5432 user=replicator

[global]
log-level-file=detail
repo1-host=repository
```

It is possible to configure a password in the `primary_conninfo` setting but using a `.pgpass` file is more flexible and secure.

pg-standby — *Configure the replication password in the* `.pgpass` *file.*

```
sudo -u postgres sh -c 'echo \
   "172.17.0.5:*:replication:replicator:jw8s0F4" \
   >> /var/lib/pgsql/.pgpass'
sudo -u postgres chmod 600 /var/lib/pgsql/.pgpass
```

Now the standby can be created with the `restore` command.

pg-standby — *Stop PostgreSQL and restore the demo standby cluster*

```
sudo systemctl stop postgresql-10.service
sudo -u postgres pgbackrest --stanza=demo --delta --type=standby restore
sudo -u postgres cat /var/lib/pgsql/10/data/recovery.conf

Output:

# Recovery settings generated by pgBackRest restore on 2021-09-08 15:34:29
primary_conninfo = 'host=172.17.0.5 port=5432 user=replicator'
restore_command = 'pgbackrest --stanza=demo archive-get %f "%p"'
standby_mode = 'on'
```

> ***NOTE:*** *The* `primary_conninfo` *setting has been written into the* `recovery.conf` *file because it was configured as a* `recovery-option` *in* `pgbackrest.conf`*. The* `--type=preserve` *option can be used with the* `restore` *to leave the existing* `recovery.conf` *file in place if that behavior is preferred.*

By default RHEL/CentOS 7-8 stores the `postgresql.conf` file in the PostgreSQL data directory. That means the change made to `postgresql.conf` was overwritten by the last restore and the

hot_standby setting must be enabled again. Other solutions to this problem are to store the postgresql.conf file elsewhere or to enable the hot_standby setting on the **pg-primary** host where it will be ignored.

pg-standby:/var/lib/pgsql/10/data/postgresql.conf — *Enable* hot_standby

```
archive_command = 'pgbackrest --stanza=demo archive-push %p'
archive_mode = on
hot_standby = on
log_filename = 'postgresql.log'
log_line_prefix = ''
max_wal_senders = 3
wal_level = replica
```

pg-standby — *Start PostgreSQL*

```
sudo systemctl start postgresql-10.service
```

The PostgreSQL log will confirm that streaming replication has started.

pg-standby — *Examine the PostgreSQL log output for log messages indicating success*

```
sudo -u postgres cat /var/lib/pgsql/10/data/log/postgresql.log

Output:

     [filtered 6 lines of output]
LOG: database system is ready to accept read only connections
LOG: restored log file "000000080000000000000001C" from archive
LOG: started streaming WAL from primary at 0/1D000000 on timeline 8
```

Now when a table is created on **pg-primary** it will appear on **pg-standby** quickly and without the need to call pg_switch_wal().

pg-primary — *Create a new table on the primary*

```
sudo -u postgres psql -c " \
   begin; \
   create table stream_table (message text); \
   insert into stream_table values ('Important Data'); \
   commit; \
   select *, current_timestamp from stream_table";

Output:

   message  |     current_timestamp
----------------+-------------------------------
 Important Data | 2021-09-08 15:34:34.213399+00
(1 row)
```

pg-standby — *Query table on the standby*

```
sudo -u postgres psql -c " \
   select *, current_timestamp from stream_table"

Output:

   message  |     current_timestamp
----------------+-------------------------------
 Important Data | 2021-09-08 15:34:34.689598+00
(1 row)
```

20 Asynchronous Archiving

Asynchronous archiving is enabled with the `archive-async` option. This option enables asynchronous operation for both the `archive-push` and `archive-get` commands.

A spool path is required. The commands will store transient data here but each command works quite a bit differently so spool path usage is described in detail in each section.

pg-primary — *Create the spool directory*

```
sudo mkdir -p -m 750 /var/spool/pgbackrest
sudo chown postgres:postgres /var/spool/pgbackrest
```

pg-standby — *Create the spool directory*

```
sudo mkdir -p -m 750 /var/spool/pgbackrest
sudo chown postgres:postgres /var/spool/pgbackrest
```

The spool path must be configured and asynchronous archiving enabled. Asynchronous archiving automatically confers some benefit by reducing the number of connections made to remote storage, but setting `process-max` can drastically improve performance by parallelizing operations. Be sure not to set `process-max` so high that it affects normal database operations.

pg-primary`:/etc/pgbackrest/pgbackrest.conf` — *Configure the spool path and asynchronous archiving*

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

[global]
archive-async=y
log-level-file=detail
repo1-host=repository
spool-path=/var/spool/pgbackrest

[global:archive-get]
process-max=2

[global:archive-push]
process-max=2
```

pg-standby`:/etc/pgbackrest/pgbackrest.conf` — *Configure the spool path and asynchronous archiving*

```
[demo]
pg1-path=/var/lib/pgsql/10/data
recovery-option=primary_conninfo=host=172.17.0.5 port=5432 user=replicator

[global]
archive-async=y
log-level-file=detail
repo1-host=repository
```

```
spool-path=/var/spool/pgbackrest

[global:archive-get]
process-max=2

[global:archive-push]
process-max=2
```

> ***NOTE:*** `process-max` *is configured using command sections so that the option is not used by backup and restore. This also allows different values for* `archive-push` *and* `archive-get`*.*

For demonstration purposes streaming replication will be broken to force PostgreSQL to get WAL using the `restore_command`.

pg-primary — *Break streaming replication by changing the replication password*

```
sudo -u postgres psql -c "alter user replicator password 'bogus'"

Output:

ALTER ROLE
```

pg-standby — *Restart standby to break connection*

```
sudo systemctl restart postgresql-10.service
```

20.1 Archive Push

The asynchronous `archive-push` command offloads WAL archiving to a separate process (or processes) to improve throughput. It works by "looking ahead" to see which WAL segments are ready to be archived beyond the request that PostgreSQL is currently making via the `archive_command`. WAL segments are transferred to the archive directly from the `pg_xlog/pg_wal` directory and success is only returned by the `archive_command` when the WAL segment has been safely stored in the archive.

The spool path holds the current status of WAL archiving. Status files written into the spool directory are typically zero length and should consume a minimal amount of space (a few MB at most) and very little IO. All the information in this directory can be recreated so it is not necessary to preserve the spool directory if the cluster is moved to new hardware.

> ***IMPORTANT:*** *In the original implementation of asynchronous archiving, WAL segments were copied to the spool directory before compression and transfer. The new implementation copies WAL directly from the* `pg_xlog` *directory. If asynchronous archiving was utilized in* `v1.12` *or prior, read the* `v1.13` *release notes carefully before upgrading.*

The `[stanza]-archive-push-async.log` file can be used to monitor the activity of the asynchronous process. A good way to test this is to quickly push a number of WAL segments.

pg-primary — *Test parallel asynchronous archiving*

```
sudo -u postgres psql -c " \
   select pg_create_restore_point('test async push'); select pg_switch_wal(); \
   select pg_create_restore_point('test async push'); select pg_switch_wal(); \
   select pg_create_restore_point('test async push'); select pg_switch_wal(); \
   select pg_create_restore_point('test async push'); select pg_switch_wal(); \
   select pg_create_restore_point('test async push'); select pg_switch_wal();"
sudo -u postgres pgbackrest --stanza=demo --log-level-console=info check

Output:

P00  INFO: check command begin 2.35: --exec-id=3762-363b0f05 --log-level-console=
   info --log-level-file=detail --log-level-stderr=off --no-log-timestamp --pg1-
   path=/var/lib/pgsql/10/data --repo1-host=repository --stanza=demo
P00  INFO: check repo1 configuration (primary)
P00  INFO: check repo1 archive for WAL (primary)
P00  INFO: WAL segment 000000080000000000000022 successfully archived to '/var/lib/
   pgbackrest/archive/demo/10-1/0000000800000000/000000080000000000000022-3
   f70bf7fa3c5291041dfc52d9c539ca7c78b4687.gz' on repo1
P00  INFO: check command end: completed successfully
```

Now the log file will contain parallel, asynchronous activity.

pg-primary — *Check results in the log*

```
sudo -u postgres cat /var/log/pgbackrest/demo-archive-push-async.log

Output:

-------------------PROCESS START-------------------
P00  INFO: archive-push:async command begin 2.35: [/var/lib/pgsql/10/data/pg_wal]
   --archive-async --exec-id=3737-398b9b22 --log-level-console=off --log-level-
   file=detail --log-level-stderr=off --no-log-timestamp --pg1-path=/var/lib/pgsql
   /10/data --process-max=2 --repo1-host=repository --spool-path=/var/spool/
   pgbackrest --stanza=demo
P00  INFO: push 1 WAL file(s) to archive: 000000080000000000000001D
P01 DETAIL: pushed WAL file '000000080000000000000001D' to the archive
P00  INFO: archive-push:async command end: completed successfully

-------------------PROCESS START-------------------
P00  INFO: archive-push:async command begin 2.35: [/var/lib/pgsql/10/data/pg_wal]
   --archive-async --exec-id=3765-bc64aba4 --log-level-console=off --log-level-
   file=detail --log-level-stderr=off --no-log-timestamp --pg1-path=/var/lib/pgsql
   /10/data --process-max=2 --repo1-host=repository --spool-path=/var/spool/
   pgbackrest --stanza=demo
P00  INFO: push 5 WAL file(s) to archive: 000000080000000000000001E
   ...000000080000000000000022
P02 DETAIL: pushed WAL file '000000080000000000000001F' to the archive
P01 DETAIL: pushed WAL file '000000080000000000000001E' to the archive
```

```
P02 DETAIL: pushed WAL file '000000080000000000000020' to the archive
P01 DETAIL: pushed WAL file '000000080000000000000021' to the archive
P02 DETAIL: pushed WAL file '000000080000000000000022' to the archive
P00  INFO: archive-push:async command end: completed successfully
```

20.2 Archive Get

The asynchronous `archive-get` command maintains a local queue of WAL to improve throughput. If a WAL segment is not found in the queue it is fetched from the repository along with enough consecutive WAL to fill the queue. The maximum size of the queue is defined by `archive-get-queue-max`. Whenever the queue is less than half full more WAL will be fetched to fill it.

Asynchronous operation is most useful in environments that generate a lot of WAL or have a high latency connection to the repository storage (i.e., `S3` or other object stores). In the case of a high latency connection it may be a good idea to increase `process-max`.

The `[stanza]-archive-get-async.log` file can be used to monitor the activity of the asynchronous process.

pg-standby — *Check results in the log*

```
sudo -u postgres cat /var/log/pgbackrest/demo-archive-get-async.log

Output:

------------------PROCESS START------------------
P00  INFO: archive-get:async command begin 2.35: [000000080000000000000001B,
   000000080000000000000001C, 000000080000000000000001D, 000000080000000000000001E,
   000000080000000000000001F, 000000080000000000000020, 000000080000000000000021,
   000000080000000000000022] --archive-async --exec-id=1168-bf79dc39 --log-level-
   console=off --log-level-file=detail --log-level-stderr=off --no-log-timestamp
   --pg1-path=/var/lib/pgsql/10/data --process-max=2 --repo1-host=repository --
   spool-path=/var/spool/pgbackrest --stanza=demo
P00  INFO: get 8 WAL file(s) from archive: 000000080000000000000001B
   ...000000080000000000000022
P02 DETAIL: found 000000080000000000000001C in the repo1: 10-1 archive
P01 DETAIL: found 000000080000000000000001B in the repo1: 10-1 archive
P00 DETAIL: unable to find 000000080000000000000001D in the archive
P00  INFO: archive-get:async command end: completed successfully
     [filtered 14 lines of output]
P00  INFO: archive-get:async command begin 2.35: [000000080000000000000001D,
   000000080000000000000001E, 000000080000000000000001F, 000000080000000000000020,
   000000080000000000000021, 000000080000000000000022, 000000080000000000000023,
   000000080000000000000024] --archive-async --exec-id=1190-4f33723c --log-level-
   console=off --log-level-file=detail --log-level-stderr=off --no-log-timestamp
   --pg1-path=/var/lib/pgsql/10/data --process-max=2 --repo1-host=repository --
   spool-path=/var/spool/pgbackrest --stanza=demo
```

```
P00  INFO: get 8 WAL file(s) from archive: 000000080000000000000001D
   ...000000080000000000000024
P01 DETAIL: found 000000080000000000000001D in the repo1: 10-1 archive
P02 DETAIL: found 000000080000000000000001E in the repo1: 10-1 archive
P01 DETAIL: found 000000080000000000000001F in the repo1: 10-1 archive
P02 DETAIL: found 000000080000000000000020 in the repo1: 10-1 archive
P01 DETAIL: found 000000080000000000000021 in the repo1: 10-1 archive
P02 DETAIL: found 000000080000000000000022 in the repo1: 10-1 archive
P00 DETAIL: unable to find 000000080000000000000023 in the archive
P00  INFO: archive-get:async command end: completed successfully
     [filtered 11 lines of output]
```

pg-primary — *Fix streaming replication by changing the replication password*

```
sudo -u postgres psql -c "alter user replicator password 'jw8s0F4'"

Output:

ALTER ROLE
```

21 Backup from a Standby

pgBackRest can perform backups on a standby instead of the primary. Standby backups require the **pg-standby** host to be configured and the `backup-standby` option enabled. If more than one standby is configured then the first running standby found will be used for the backup.

repository*:/etc/pgbackrest/pgbackrest.conf* — *Configure* `pg2-host/pg2-host-user` *and* `pg2-path`

```
[demo]
pg1-host=pg-primary
pg1-path=/var/lib/pgsql/10/data
pg2-host=pg-standby
pg2-path=/var/lib/pgsql/10/data

[global]
backup-standby=y
process-max=3
repo1-path=/var/lib/pgbackrest
repo1-retention-full=2
start-fast=y
```

Both the primary and standby databases are required to perform the backup, though the vast majority of the files will be copied from the standby to reduce load on the primary. The database hosts can be configured in any order. pgBackRest will automatically determine which is the primary and which is the standby.

repository — *Backup the demo cluster from* **pg2**

```
sudo -u pgbackrest pgbackrest --stanza=demo --log-level-console=detail backup

Output:

     [filtered 2 lines of output]
P00  INFO: execute non-exclusive pg_start_backup(): backup begins after the
   requested immediate checkpoint completes
P00  INFO: backup start archive = 000000080000000000000024, lsn = 0/24000028
P00  INFO: wait for replay on the standby to reach 0/24000028
P00  INFO: replay on the standby reached 0/24000028
P01 DETAIL: backup file pg-primary:/var/lib/pgsql/10/data/global/pg_control (8KB,
   0%) checksum a40e2f5c3fd23352182bf3f415d9d1a68094af87
P01 DETAIL: backup file pg-primary:/var/lib/pgsql/10/data/log/postgresql.log (7.0
   KB, 0%) checksum 13e3641a0ff2f7fa8e0ce904c42d81058e271e3f
P01 DETAIL: backup file pg-primary:/var/lib/pgsql/10/data/pg_hba.conf (4.2KB, 0%)
   checksum 12abee43e7eabfb3ff6239f3fc9bc3598293557d
P01 DETAIL: backup file pg-primary:/var/lib/pgsql/10/data/current_logfiles (26B,
   0%) checksum 78a9f5c10960f0d91fcd313937469824861795a2
P01 DETAIL: backup file pg-primary:/var/lib/pgsql/10/data/pg_logical/
   replorigin_checkpoint (8B, 0%) checksum 347
```

```
   fc8f2df71bd4436e38bd1516ccd7ea0d46532
P02 DETAIL: backup file pg-standby:/var/lib/pgsql/10/data/base/12953/2608 (440KB,
   19%) checksum 07508eb67ab49ffd3db245478ab038c61dca43f7
P03 DETAIL: backup file pg-standby:/var/lib/pgsql/10/data/base/12953/1249 (392KB,
   37%) checksum 8807d2fe68061f264365e13275373b69bc70e7d3
     [filtered 1254 lines of output]
```

This incremental backup shows that most of the files are copied from the **pg-standby** host and only a few are copied from the **pg-primary** host.

pgBackRest creates a standby backup that is identical to a backup performed on the primary. It does this by starting/stopping the backup on the **pg-primary** host, copying only files that are replicated from the **pg-standby** host, then copying the remaining few files from the **pg-primary** host. This means that logs and statistics from the primary database will be included in the backup.

22 Upgrading PostgreSQL

Immediately after upgrading PostgreSQL to a newer major version, the `pg-path` for all pgBackRest configurations must be set to the new database location and the `stanza-upgrade` command run. If there is more than one repository configured on the host, the stanza will be created on each. If the database is offline use the `--no-online` option.

The following instructions are not meant to be a comprehensive guide for upgrading PostgreSQL, rather they outline the general process for upgrading a primary and standby with the intent of demonstrating the steps required to reconfigure pgBackRest. It is recommended that a backup be taken prior to upgrading.

pg-primary — *Stop old cluster*

```
sudo systemctl stop postgresql-10.service
```

Stop the old cluster on the standby since it will be restored from the newly upgraded cluster.

pg-standby — *Stop old cluster*

```
sudo systemctl stop postgresql-10.service
```

Create the new cluster and perform upgrade.

pg-primary — *Create new cluster and perform the upgrade*

```
sudo -u postgres /usr/pgsql-11/bin/initdb \
   -D /var/lib/pgsql/11/data -k -A peer
sudo -u postgres sh -c 'cd /var/lib/pgsql && \
   /usr/pgsql-11/bin/pg_upgrade \
   --old-bindir=/usr/pgsql-10/bin \
   --new-bindir=/usr/pgsql-11/bin \
   --old-datadir=/var/lib/pgsql/10/data \
   --new-datadir=/var/lib/pgsql/11/data \
   --old-options=" -c config_file=/var/lib/pgsql/10/data/postgresql.conf" \
   --new-options=" -c config_file=/var/lib/pgsql/11/data/postgresql.conf"'

Output:

     [filtered 72 lines of output]
Checking for extension updates                ok

Upgrade Complete
----------------
Optimizer statistics are not transferred by pg_upgrade so,
     [filtered 4 lines of output]
```

Configure the new cluster settings and port.

pg-primary`:/var/lib/pgsql/11/data/postgresql.conf` — *Configure PostgreSQL*

```
archive_command = 'pgbackrest --stanza=demo archive-push %p'
archive_mode = on
listen_addresses = '*'
log_line_prefix = ''
max_wal_senders = 3
port = 5432
wal_level = replica
```

Update the pgBackRest configuration on all systems to point to the new cluster.

pg-primary*:/etc/pgbackrest/pgbackrest.conf — Upgrade the* pg1-path

```
[demo]
pg1-path=/var/lib/pgsql/11/data

[global]
archive-async=y
log-level-file=detail
repo1-host=repository
spool-path=/var/spool/pgbackrest

[global:archive-get]
process-max=2

[global:archive-push]
process-max=2
```

pg-standby*:/etc/pgbackrest/pgbackrest.conf — Upgrade the* pg-path

```
[demo]
pg1-path=/var/lib/pgsql/11/data
recovery-option=primary_conninfo=host=172.17.0.5 port=5432 user=replicator

[global]
archive-async=y
log-level-file=detail
repo1-host=repository
spool-path=/var/spool/pgbackrest

[global:archive-get]
process-max=2

[global:archive-push]
process-max=2
```

repository*:/etc/pgbackrest/pgbackrest.conf — Upgrade* pg1-path *and* pg2-path*, disable backup from standby*

```
[demo]
pg1-host=pg-primary
```

```
pg1-path=/var/lib/pgsql/11/data
pg2-host=pg-standby
pg2-path=/var/lib/pgsql/11/data

[global]
backup-standby=n
process-max=3
repo1-path=/var/lib/pgbackrest
repo1-retention-full=2
start-fast=y
```

pg-primary — *Copy hba configuration*

```
sudo cp /var/lib/pgsql/10/data/pg_hba.conf \
   /var/lib/pgsql/11/data/pg_hba.conf
```

Before starting the new cluster, the `stanza-upgrade` command must be run.

pg-primary — *Upgrade the stanza*

```
sudo -u postgres pgbackrest --stanza=demo --no-online \
   --log-level-console=info stanza-upgrade

Output:

P00  INFO: stanza-upgrade command begin 2.35: --exec-id=4179-62ff994c --log-level-
   console=info --log-level-file=detail --log-level-stderr=off --no-log-timestamp
   --no-online --pg1-path=/var/lib/pgsql/11/data --repo1-host=repository --stanza=
   demo
P00  INFO: stanza-upgrade for stanza 'demo' on repo1
P00  INFO: stanza-upgrade command end: completed successfully
```

Start the new cluster and confirm it is successfully installed.

pg-primary — *Start new cluster*

```
sudo systemctl start postgresql-11.service
```

Test configuration using the `check` command.

pg-primary — *Check configuration*

```
sudo -u postgres systemctl status postgresql-11.service

Output:

   postgresql-11.service - PostgreSQL 11 database server
  Loaded: loaded (/usr/lib/systemd/system/postgresql-11.service; disabled; vendor
     preset: disabled)
  Active: active (running) since Wed 2021-09-08 15:35:11 UTC; 321ms ago
    Docs: https://www.postgresql.org/docs/11/static/
```

```
 Process: 4200 ExecStartPre=/usr/pgsql-11/bin/postgresql-11-check-db-dir ${PGDATA}
     (code=exited, status=0/SUCCESS)
Main PID: 4205 (postmaster)
  CGroup: /docker/ad15be547e0a403d58873e292ca90d94c821951a9ca28010aa234a38d167034a
     /system.slice/postgresql-11.service
          4205   /usr/pgsql-11/bin/postmaster -D /var/lib/pgsql/11/data/
          4206   postgres: logger
          4208   postgres: checkpointer
          4209   postgres: background writer
          4210   postgres: walwriter
          4211   postgres: autovacuum launcher
          4212   postgres: archiver
          4213   postgres: stats collector
          4214   postgres: logical replication launcher
sudo -u postgres pgbackrest --stanza=demo check
```

Remove the old cluster.

pg-primary — *Remove old cluster*

```
sudo rm -rf /var/lib/pgsql/10/data
```

Install the new PostgreSQL binaries on the standby and create the cluster.

pg-standby — *Remove old cluster and create the new cluster*

```
sudo rm -rf /var/lib/pgsql/10/data
sudo -u postgres mkdir -p -m 700 /usr/pgsql-11/bin
```

Run the `check` on the repository host. The warning regarding the standby being down is expected since the standby cluster is down. Running this command demonstrates that the repository server is aware of the standby and is configured properly for the primary server.

repository — *Check configuration*

```
sudo -u pgbackrest pgbackrest --stanza=demo check

Output:

P00  WARN: unable to check pg-2: [DbConnectError] raised from remote-0 protocol on
   'pg-standby': unable to connect to 'dbname='postgres' port=5432': could not
   connect to server: No such file or directory
            Is the server running locally and accepting
            connections on Unix domain socket "/var/run/postgresql/.s.PGSQL.5432"?
```

Run a full backup on the new cluster and then restore the standby from the backup. The backup type will automatically be changed to `full` if `incr` or `diff` is requested.

repository — *Run a full backup*

```
sudo -u pgbackrest pgbackrest --stanza=demo --type=full backup
```

pg-standby — *Restore the demo standby cluster*

```
sudo -u postgres pgbackrest --stanza=demo --type=standby restore
```

pg-standby*:/var/lib/pgsql/11/data/postgresql.conf* — *Configure PostgreSQL*

```
hot_standby = on
```

pg-standby — *Start PostgreSQL and check the pgBackRest configuration*

```
sudo systemctl start postgresql-11.service
sudo -u postgres pgbackrest --stanza=demo check
```

Backup from standby can be enabled now that the standby is restored.

repository*:/etc/pgbackrest/pgbackrest.conf* — *Reenable backup from standby*

```
[demo]
pg1-host=pg-primary
pg1-path=/var/lib/pgsql/11/data
pg2-host=pg-standby
pg2-path=/var/lib/pgsql/11/data

[global]
backup-standby=y
process-max=3
repo1-path=/var/lib/pgbackrest
repo1-retention-full=2
start-fast=y
```