

pgBackRest User Guide

Debian & Ubuntu / PostgreSQL 12-13

[\[Home\]](#) [\[User Guides\]](#) [\[Releases\]](#) [\[Configuration\]](#) [\[Commands\]](#) [\[FAQ\]](#) [\[Metrics\]](#)

Table of Contents

- 1 [Introduction](#)
- 2 [Concepts](#)
 - 2.1 [Backup](#)
 - 2.2 [Restore](#)
 - 2.3 [Write Ahead Log \(WAL\)](#)
 - 2.4 [Encryption](#)
- 3 [Upgrading pgBackRest](#)
 - 3.1 [Upgrading pgBackRest from v1 to v2](#)
- 4 [Build](#)
- 5 [Installation](#)
- 6 [Quick Start](#)
 - 6.1 [Setup Demo Cluster](#)
 - 6.2 [Configure Cluster Stanza](#)
 - 6.3 [Create the Repository](#)
 - 6.4 [Configure Archiving](#)
 - 6.5 [Configure Retention](#)
 - 6.6 [Configure Repository Encryption](#)
 - 6.7 [Create the Stanza](#)
 - 6.8 [Check the Configuration](#)
 - 6.9 [Perform a Backup](#)
 - 6.10 [Schedule a Backup](#)
 - 6.11 [Backup Information](#)
 - 6.12 [Restore a Backup](#)
- 7 [Backup](#)
 - 7.1 [Fast Start Option](#)
 - 7.2 [Archive Timeout](#)
- 8 [Monitoring](#)
 - 8.1 [In PostgreSQL](#)
 - 8.2 [Using jq](#)
- 9 [Retention](#)
 - 9.1 [Full Backup Retention](#)

- 9.2 [Differential Backup Retention](#)
- 9.3 [Archive Retention](#)
- 10 [Restore](#)
 - 10.1 [File Ownership](#)
 - 10.2 [Delta Option](#)
 - 10.3 [Restore Selected Databases](#)
- 11 [Point-in-Time Recovery](#)
- 12 [Azure-Compatible Object Store Support](#)
- 13 [S3-Compatible Object Store Support](#)
- 14 [Delete a Stanza](#)
- 15 [Dedicated Repository Host](#)
 - 15.1 [Installation](#)
 - 15.2 [Setup Passwordless SSH](#)
 - 15.3 [Configuration](#)
 - 15.4 [Perform a Backup](#)
 - 15.5 [Restore a Backup](#)
- 16 [Parallel Backup / Restore](#)
- 17 [Starting and Stopping](#)
- 18 [Replication](#)
 - 18.1 [Installation](#)
 - 18.2 [Setup Passwordless SSH](#)
 - 18.3 [Hot Standby](#)
 - 18.4 [Streaming Replication](#)
- 19 [Asynchronous Archiving](#)
 - 19.1 [Archive Push](#)
 - 19.2 [Archive Get](#)
- 20 [Backup from a Standby](#)
- 21 [Upgrading PostgreSQL](#)

1 Introduction

This user guide is intended to be followed sequentially from beginning to end — each section depends on the last. For example, the [Backup](#) 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 Debian/Ubuntu and PostgreSQL 12-13, 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.conf` 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

Debian/Ubuntu packages for pgBackRest are available at apt.postgresql.org. If they are not provided for your distribution/version it is 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 2.30 of pgBackRest to pre-created /build path

```
$ wget -q -O - \
    https://github.com/pgbackrest/pgbackrest/archive/release
/2.30.tar.gz | \
    tar zx -C /build
```

build ⇒ Install build dependencies

```
$ sudo apt-get install make gcc libpq-dev libssl-dev libxml2-dev pkg-
config \
    liblz4-dev libzstd-dev libbz2-dev libz-dev
```

build ⇒ Configure and compile pgBackRest

```
$ cd /build/pgbackrest-release-2.30/src && ./configure && make
```

5 Installation

A new host named `pg1` 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.

build ⇒ Install dependencies

```
$ sudo apt-get install postgresql-client libxml2
```

pg-primary ⇒ Copy `pgBackRest` binary from build host

```
$ sudo scp build:/build/pgbackrest-release-2.30/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
pgBackRest 2.30 - 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.
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/lib/postgresql/12/bin/initdb \
-D /var/lib/postgresql/12/demo -k -A peer
$ sudo pg_createcluster 12 demo

Configuring already existing cluster (configuration: /etc/postgresql
/12/demo, data: /var/lib/postgresql/12/demo, owner: 106:110)
Ver Cluster Port Status Owner    Data directory          Log file
12  demo    5432 down   postgres /var/lib/postgresql/12/demo /var/log
/postgresql/postgresql-12-demo.log
```

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:/etc/postgresql/12/demo/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:/etc/postgresql/12/demo/postgresql.conf ⇒ Set log_line_prefix
listen_addresses = '*'
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 Debian/Ubuntu stores clusters in `/var/lib/postgresql/[version]/[cluster]` 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/postgresql/12/demo
```

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 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'
pgBackRest 2.30 - '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/postgresql/12/demo

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

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:/etc/postgresql/12/demo/postgresql.conf ⇒ Configure archive settings

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

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 pg_ctlcluster 12 demo restart
```

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/postgresql/12/demo

[global]
repol-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/postgresql/12/demo

[global]
repol-path=/var/lib/pgbackrest
repol-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/postgresql/12/demo
```

```
[global]
repol-cipher-
pass=zWaf6XtpjIVZC5444yXB+cgFDF17MxGlgkZSaoPvTGirhPygu4jOKOXf9L04vjf0
repol-cipher-type=aes-256-cbc
repol-path=/var/lib/pgbackrest
repol-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 on the host where the repository is located 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
P00 INFO: stanza-create command begin 2.30: --log-level-console=info
--log-level-stderr=off --no-log-timestamp --pg1-path=/var
/lib/postgresql/12/demo --repol-cipher-pass= --repol-cipher-type=aes-
256-cbc --repol-path=/var/lib/pgbackrest --stanza=demo
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. 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 database or the 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 `>= 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
P00 INFO: check command begin 2.30: --log-level-console=info --log-
```

```

level-stderr=off --no-log-timestamp --pgl-path=/var/lib/postgresql
/12/demo --repol-cipher-pass= --repol-cipher-type=aes-256-cbc --repol-
path=/var/lib/pgbackrest --stanza=demo
P00 INFO: WAL segment 0000000100000000000000001 successfully archived
to '/var/lib/pgbackrest/archive/demo/12-1/0000000100000000
/0000000100000000000000001-94df38a6c71b0ba12053726ae869356a4df6e3d2.gz'
P00 INFO: check command end: completed successfully

```

6.9 Perform a Backup

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
P00 INFO: backup command begin 2.30: --log-level-console=info --log-
level-stderr=off --no-log-timestamp --pgl-path=/var/lib/postgresql
/12/demo --repol-cipher-pass= --repol-cipher-type=aes-256-cbc --repol-
path=/var/lib/pgbackrest --repol-retention-full=2 --stanza=demo
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 next regular checkpoint completes
P00 INFO: backup start archive = 0000000100000000000000002, lsn =
0/2000028
    [filtered 973 lines of output]
P01 INFO: backup file /var/lib/postgresql/12/demo/base/1/13201 (0B,
100%)
P01 INFO: backup file /var/lib/postgresql/12/demo/base/1/13196 (0B,
100%)
P00 INFO: full backup size = 23.4MB
P00 INFO: execute non-exclusive pg_stop_backup() and wait for all WAL
segments to archive
P00 INFO: backup stop archive = 0000000100000000000000002, lsn =
0/2000138
    [filtered 4 lines of output]

```

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
    [filtered 4 lines of output]
P01 INFO: backup file /var/lib/postgresql/12/demo/global/pg_control
(8KB, 99%) checksum cec5a5adcdfe5dc47453928094dacdd6934cc91c
P01 INFO: backup file /var/lib/postgresql/12/demo/pg_logical
/replorigin_checkpoint (8B, 100%) checksum
347fc8f2df71bd4436e38bd1516ccd7ea0d46532
P00 INFO: diff backup size = 8KB
P00 INFO: execute non-exclusive pg_stop_backup() and wait for all WAL

```

```

segments to archive
P00 INFO: backup stop archive = 00000001000000000000000004, lsn =
0/4000100
[filtered 4 lines of output]

```

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`.

More information about the backup command can be found in the [Backup](#) section.

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.

```

#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

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

db (current)
wal archive min/max (12-1):
000000010000000000000001/000000010000000000000004
full backup: 20201005-142231F
timestamp start/stop: 2020-10-05 14:22:31 / 2020-10-05
14:22:47 wal start/stop: 000000010000000000000002 /
000000010000000000000002
database size: 23.4MB, backup size: 23.4MB
repository size: 2.8MB, repository backup size: 2.8MB
diff backup: 20201005-142231F_20201005-142249D
timestamp start/stop: 2020-10-05 14:22:49 / 2020-10-05
14:22:51 wal start/stop: 000000010000000000000004 /

```

```
00000001000000000000000004
    database size: 23.4MB, backup size: 8.3KB
    repository size: 2.8MB, repository backup size: 496B
    backup reference list: 20201005-142231F
```

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. The 'wal archive min/max' shows the minimum and maximum WAL currently stored in the archive. 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 'backup size' is the amount of data actually backed up (these will be the same for full backups). The 'repository size' includes all the files from this backup and any referenced backups that are required to restore the database while 'repository 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 pg_ctlcluster 12 demo stop
$ sudo -u postgres rm /var/lib/postgresql/12/demo/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 pg_ctlcluster 12 demo start
Error: /usr/lib/postgresql/12/bin/pg_ctl /usr/lib/postgresql/12/bin
/pg_ctl start -D /var/lib/postgresql/12/demo -l /var/log/postgresql
/postgresql-12-demo.log -s -o -c config_file="/etc/postgresql/12/demo
/postgresql.conf" exited with status 1:
postgres: could not find the database system
Expected to find it in the directory "/var/lib/postgresql/12/demo",
but could not open file "/var/lib/postgresql/12/demo/global
/pg_control": No such file or directory
Examine the log output.
```

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/postgresql/12/demo -mindepth 1 -delete
```

pg-primary ⇒ Restore the demo cluster and start PostgreSQL

```
$ sudo -u postgres pgbackrest --stanza=demo restore
$ sudo pg_ctlcluster 12 demo start
```

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 Backup

The Backup section introduces additional backup command features.

7.1 Fast Start Option

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.

pg-primary ⇒ Incremental backup of the demo cluster with the regularly scheduled checkpoint

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

P00 INFO: backup command begin 2.30: --log-level-console=info --log-
level-stderr=off --no-log-timestamp --pg1-path=/var/lib/postgresql
/12/demo --repol-cipher-pass= --repol-cipher-type=aes-256-cbc --repol-
path=/var/lib/pgbackrest --repol-retention-full=2 --stanza=demo
--type=incr
P00 INFO: last backup label = 20201005-142231F_20201005-142249D,
version = 2.30
P00 INFO: execute non-exclusive pg_start_backup(): backup begins
after the next regular checkpoint completes
P00 INFO: backup start archive = 00000002000000000000000006, lsn =
0/6000028
P00 WARN: a timeline switch has occurred since the
20201005-142231F_20201005-142249D backup, enabling delta checksum
[filtered 10 lines of output]
```

When `--start-fast` is passed on the command-line or `start-fast=y` is set in `/etc/pgbackrest/pgbackrest.conf` an immediate checkpoint is requested and the backup will start more quickly. This is convenient for testing and for ad-hoc backups. For instance, if a backup is being taken at the beginning of a release window it makes no sense to wait for a checkpoint. Since regularly scheduled backups generally only happen once per day it is unlikely that enabling the `start-fast` in `/etc/pgbackrest/pgbackrest.conf` will negatively affect performance, however for high-volume transactional systems you may want to pass `--start-fast` on the command-line instead. Alternately, it is possible to override the setting in the configuration file by passing `--no-start-fast` on the command-line.

pg-primary:/etc/pgbackrest/pgbackrest.conf ⇒ Enable the `start-fast` option

```
[demo]
pg1-path=/var/lib/postgresql/12/demo

[global]
repol-cipher-
pass=zWaf6XtpjIVZC5444yXB+cgFDF17MxGlGkZSaoPvTGirhPygu4jOKOXf9L04vjfO
repol-cipher-type=aes-256-cbc
repol-path=/var/lib/pgbackrest
repol-retention-full=2
start-fast=y

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

pg-primary ⇒ Incremental backup of the demo cluster with an immediate checkpoint

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

P00 INFO: backup command begin 2.30: --log-level-console=info --log-
```

```

level-stderr=off --no-log-timestamp --pg1-path=/var/lib/postgresql
/12/demo --repol-cipher-pass= --repol-cipher-type=aes-256-cbc --repol-
path=/var/lib/pgbackrest --repol-retention-full=2 --stanza=demo
--start-fast --type=incr
P00 INFO: last backup label = 20201005-142231F_20201005-142312I,
version = 2.30
P00 INFO: execute non-exclusive pg_start_backup(): backup begins
after the requested immediate checkpoint completes
P00 INFO: backup start archive = 00000002000000000000000007, lsn =
0/7000028
P01 INFO: backup file /var/lib/postgresql/12/demo/global/pg_control
(8KB, 99%) checksum 33632560c26baa2ef3c887521ab0d73a0ae09f61
[filtered 8 lines of output]

```

7.2 Archive Timeout

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.

8 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.

8.1 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/postgresql/pgbackrest/doc/example/pgsql-pgbackrest-
info.sql
-- 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/postgresql/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/postgresql/pgbackrest/doc/example/pgsql-pgbackrest-
query.sql
-- 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/postgresql/pgbackrest/doc/example/pgsql-pgbackrest-
query.sql

```

name	last_successful_backup	last_archived_wal
"demo"	2020-10-05 14:23:18+00	00000002000000000000000007

(1 row)

8.2 Using jq

jq is a command-line utility that can easily extract data from JSON.

pg-primary ⇒ Install jq utility

```
$ sudo apt-get install jq
```

Now jq can be used to query the last successful backup time for a stanza.

pg-primary ⇒ Query last successful backup time

```
$ sudo -u postgres pgbackrest --output=json --stanza=demo info | \
jq '[0] | .backup[-1] | .timestamp.stop'
1601907798
```

Or the last archived WAL.

pg-primary ⇒ Query last archived WAL

```
$ sudo -u postgres pgbackrest --output=json --stanza=demo info | \
jq '[0] | .archive[-1] | .max'
"00000002000000000000000000000007"
```

NOTE: This syntax requires jq v1.5.

NOTE: jq may round large numbers such as system identifiers. Test your queries carefully.

9 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.

9.1 Full Backup Retention

The `repol-retention-full-type` determines how the option `repol-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 `repol-retention-full-type=count` and `repol-retention-full=2` then there will be three full backups stored before the oldest one is expired, or if `repol-retention-full-type=time` and `repol-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
repol-retention-full		
[demo]		
pg1-path=/var/lib/postgresql/12/demo		
[global]		
repol-cipher-		
pass=zWaf6XtpjIVZC5444yXB+cgFDF17MxGlGkZSaoPvTGirhPygu4jOKOXf9L04vjfO		
repol-cipher-type=aes-256-cbc		

```

repol-path=/var/lib/pgbackrest
repol-retention-full=2
start-fast=y

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

```

Backup `repol-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

[filtered 984 lines of output]
P00 INFO: backup command end: completed successfully
P00 INFO: expire command begin 2.30: --log-level-console=detail
--log-level-stderr=off --no-log-timestamp --repol-cipher-pass= --repol-
cipher-type=aes-256-cbc --repol-path=/var/lib/pgbackrest --repol-
retention-full=2 --stanza=demo
P00 DETAIL: archive retention on backup 20201005-142231F, archiveId =
12-1, start = 00000001000000000000000000000002
P00 DETAIL: remove archive: archiveId = 12-1, start =
00000001000000000000000000000001, stop = 00000001000000000000000000000001
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

[filtered 983 lines of output]
P00 INFO: backup command end: completed successfully
P00 INFO: expire command begin 2.30: --log-level-console=info --log-
level-stderr=off --no-log-timestamp --repol-cipher-pass= --repol-
cipher-type=aes-256-cbc --repol-path=/var/lib/pgbackrest --repol-
retention-full=2 --stanza=demo
P00 INFO: expire full backup set: 20201005-142231F,
20201005-142231F_20201005-142249D, 20201005-142231F_20201005-142312I,
20201005-142231F_20201005-142316I
P00 INFO: remove expired backup 20201005-142231F_20201005-142316I
P00 INFO: remove expired backup 20201005-142231F_20201005-142312I
[filtered 2 lines of output]

```

The `20201005-142231F` full backup is expired and archive retention is based on the `20201005-142327F` which is now the oldest full backup.

9.2 Differential Backup Retention

Set `repol-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
repol-retention-diff

[demo]
pg1-path=/var/lib/postgresql/12/demo

[global]
repol-cipher-
pass=zWaf6XtpjIVZC5444yXB+cgFDF17MxGlGkZSaoPvTGirhPygu4jOKOXf9LO4vjfO
repol-cipher-type=aes-256-cbc
repol-path=/var/lib/pgbackrest
repol-retention-diff=1
repol-retention-full=2
start-fast=y

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

Backup `repol-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
[filtered 12 lines of output]
P00 INFO: backup command end: completed successfully
P00 INFO: expire command begin 2.30: --log-level-console=info --log-
level-stderr=off --no-log-timestamp --repol-cipher-pass= --repol-
cipher-type=aes-256-cbc --repol-path=/var/lib/pgbackrest --repol-
retention-diff=1 --repol-retention-full=2 --stanza=demo
P00 INFO: expire diff backup set: 20201005-142345F_20201005-142407D,
20201005-142345F_20201005-142410I
P00 INFO: remove expired backup 20201005-142345F_20201005-142410I
P00 INFO: remove expired backup 20201005-142345F_20201005-142407D
```

9.3 Archive Retention

Although **pgBackRest** automatically removes archived WAL segments when expiring backups (the default expires WAL for full backups based on the `repol-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
repol-retention-diff

```
[demo]
pg1-path=/var/lib/postgresql/12/demo

[global]
repol-cipher-
pass=zWaf6XtpjIVZC5444yXB+cgFDF17MxGlgkZSaoPvTGirhPygu4jOKOXf9L04vjfO
repol-cipher-type=aes-256-cbc
repol-path=/var/lib/pgbackrest
repol-retention-diff=2
repol-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

    [filtered 9 lines of output]
P00 INFO: backup stop archive = 0000000200000000000000011, lsn =
0/11000100
P00 INFO: check archive for segment(s)
0000000200000000000000011:0000000200000000000000011
P00 INFO: new backup label = 20201005-142345F_20201005-142418D
P00 INFO: backup command end: completed successfully
P00 INFO: expire command begin 2.30: --log-level-console=info --log-
level-stderr=off --no-log-timestamp --repol-cipher-pass= --repol-
cipher-type=aes-256-cbc --repol-path=/var/lib/pgbackrest --repol-
retention-diff=2 --repol-retention-full=2 --stanza=demo
```

pg-primary ⇒ Expire archive

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

P00 INFO: expire command begin 2.30: --log-level-console=detail
--log-level-stderr=off --no-log-timestamp --repol-cipher-pass= --repol-
cipher-type=aes-256-cbc --repol-path=/var/lib/pgbackrest --repol-
retention-archive=1 --repol-retention-archive-type=diff --repol-
retention-diff=2 --repol-retention-full=2 --stanza=demo
P00 DETAIL: archive retention on backup 20201005-142327F, archiveId =
12-1, start = 0000000200000000000000009, stop = 000000020000000000000009
P00 DETAIL: archive retention on backup 20201005-142345F, archiveId =
12-1, start = 000000020000000000000000A, stop = 000000020000000000000000A
P00 DETAIL: archive retention on backup
20201005-142345F_20201005-142413D, archiveId = 12-1, start =
000000020000000000000000E, stop = 000000020000000000000000E
P00 DETAIL: archive retention on backup
20201005-142345F_20201005-142418D, archiveId = 12-1, start =
0000000200000000000000011
P00 DETAIL: remove archive: archiveId = 12-1, start =
```

```

00000002000000000000000000000000B, stop = 000000020000000000000000000000D
P00 DETAIL: remove archive: archiveId = 12-1, start =
00000002000000000000000000000000F, stop = 000000020000000000000000000010
P00 INFO: expire command end: completed successfully

```

The 20201005-142345F_20201005-142413D 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 20201005-142345F_20201005-142413D but before 20201005-142345F_20201005-142418D are removed. WAL segments generated after the new backup 20201005-142345F_20201005-142418D 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.

10 Restore

The Restore section introduces additional `restore` command features.

10.1 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.

10.2 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 pg_ctlcluster 12 demo stop
$ sudo -u postgres pgbackrest --stanza=demo --delta \
    --log-level-console=detail restore
    [filtered 2 lines of output]
P00 DETAIL: check '/var/lib/postgresql/12/demo' 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/postgresql
/12/demo'
P00 DETAIL: remove invalid file '/var/lib/postgresql/12/demo
/backup_label.old'
P00 DETAIL: remove invalid file '/var/lib/postgresql/12/demo/base/1
/pg_internal.init'
    [filtered 815 lines of output]
P01 DETAIL: restore file /var/lib/postgresql/12/demo/base/13358
/PG_VERSION - exists and matches backup (3B, 99%) checksum
ad552e6dc057d1d825bf49df79d6b98eba846ebe
P01 DETAIL: restore file /var/lib/postgresql/12/demo/base/1/PG_VERSION
- exists and matches backup (3B, 99%) checksum
ad552e6dc057d1d825bf49df79d6b98eba846ebe
P01 DETAIL: restore file /var/lib/postgresql/12/demo/PG_VERSION -
exists and matches backup (3B, 100%) checksum
ad552e6dc057d1d825bf49df79d6b98eba846ebe
P01 DETAIL: restore file /var/lib/postgresql/12/demo/global/6100 -
exists and is zero size (0B, 100%)
P01 DETAIL: restore file /var/lib/postgresql/12/demo/global/6000 -
exists and is zero size (0B, 100%)
    [filtered 201 lines of output]
```

pg-primary ⇒ Restart PostgreSQL

```
$ sudo pg_ctlcluster 12 demo start
```

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

pg-primary ⇒ Create two test databases and perform a backup

```
$ sudo -u postgres psql -c "create database test1;"
CREATE DATABASE
$ sudo -u postgres psql -c "create database test2;"
```

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

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

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/postgresql/12/demo/base/24576
7.8M    /var/lib/postgresql/12/demo/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=20201005-142345F_20201005-142431I info
[filtered 11 lines of output]
repository size: 4.7MB, repository backup size: 1.9MB
backup reference list: 20201005-142345F,
20201005-142345F_20201005-142418D
database list: postgres (13359), test1 (24576), test2
(24577)
```

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

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

```
$ sudo pg_ctlcluster 12 demo stop
$ sudo -u postgres pgbackrest --stanza=demo --delta \
--db-include=test2 restore
$ sudo pg_ctlcluster 12 demo start
```

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
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
psql: error: could not connect to server: 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/postgresql/12/demo/base/24576
176K    /var/lib/postgresql/12/demo/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;"
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;"

```

oid	datname
1	template1
13358	template0
13359	postgres
24577	test2

```

(4 rows)

```

11 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 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;"
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"
2020-10-05 14:25:05.993548+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;"
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 2020-10-05 14:25:05.993548+00, and display postgresql.auto.conf

```
$ sudo pg_ctlcluster 12 demo stop
$ sudo -u postgres pgbackrest --stanza=demo --delta \
    --type=time "--target=2020-10-05 14:25:05.993548+00" \
    --target-action=promote restore
$ sudo -u postgres cat /var/lib/postgresql/12/demo/postgresql.auto.conf
    [filtered 12 lines of output]
# Recovery settings generated by pgBackRest restore on 2020-10-05
14:25:09
restore_command = 'pgbackrest --stanza=demo archive-get %f "%p"'
recovery_target_time = '2020-10-05 14:25:05.993548+00'
recovery_target_action = 'promote'
```

pgBackRest has automatically generated the recovery settings in postgresql.auto.conf so PostgreSQL can be started immediately. 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 pg_ctlcluster 12 demo start
$ sudo -u postgres psql -c "select * from important_table"
    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/log/postgresql/postgresql-12-demo.log
    [filtered 3 lines of output]
LOG:  listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"
LOG:  database system was interrupted; last known up at 2020-10-05
14:24:57 UTC
LOG:  starting point-in-time recovery to 2020-10-05 14:25:05.993548+00
LOG:  restored log file "00000004.history" from archive
LOG:  restored log file "00000004000000000000000016" from archive
    [filtered 2 lines of output]
LOG:  database system is ready to accept read only connections
LOG:  restored log file "00000004000000000000000017" from archive
LOG:  recovery stopping before commit of transaction 495, time
2020-10-05 14:25:07.613445+00
LOG:  redo done at 0/17019DD8
LOG:  last completed transaction was at log time 2020-10-05
14:25:04.492513+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;"
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

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

db (current)
  wal archive min/max (12-1):
00000002000000000000000009/00000005000000000000000018

    full backup: 20201005-142327F
      timestamp start/stop: 2020-10-05 14:23:27 / 2020-10-05
14:23:44
      wal start/stop: 00000002000000000000000009 /
00000002000000000000000009
      database size: 23.4MB, backup size: 23.4MB
      repository size: 2.8MB, repository backup size: 2.8MB

    full backup: 20201005-142345F
      timestamp start/stop: 2020-10-05 14:23:45 / 2020-10-05
14:24:04
      wal start/stop: 0000000200000000000000000A /
0000000200000000000000000A
      database size: 23.4MB, backup size: 23.4MB
      repository size: 2.8MB, repository backup size: 2.8MB

    diff backup: 20201005-142345F_20201005-142418D
      timestamp start/stop: 2020-10-05 14:24:18 / 2020-10-05
14:24:20
      wal start/stop: 00000002000000000000000011 /
00000002000000000000000011
      database size: 23.4MB, backup size: 112.3KB
      repository size: 2.8MB, repository backup size: 13.3KB
      backup reference list: 20201005-142345F

    incr backup: 20201005-142345F_20201005-142431I
      timestamp start/stop: 2020-10-05 14:24:31 / 2020-10-05
14:24:43
      wal start/stop: 00000003000000000000000013 /
00000003000000000000000013
```

```

        database size: 38.6MB, backup size: 15.7MB
        repository size: 4.7MB, repository backup size: 1.9MB
        backup reference list: 20201005-142345F,
20201005-142345F_20201005-142418D

        diff backup: 20201005-142345F_20201005-142457D
        timestamp start/stop: 2020-10-05 14:24:57 / 2020-10-05
14:25:03
        wal start/stop: 00000004000000000000000016 /
00000004000000000000000016
        database size: 31MB, backup size: 8.2MB
        repository size: 3.8MB, repository backup size: 1007.8KB
        backup reference list: 20201005-142345F

        incr backup: 20201005-142345F_20201005-142517I
        timestamp start/stop: 2020-10-05 14:25:17 / 2020-10-05
14:25:18
        wal start/stop: 00000005000000000000000018 /
00000005000000000000000018
        database size: 31MB, backup size: 2.2MB
        repository size: 3.8MB, repository backup size: 234.0KB
        backup reference list: 20201005-142345F,
20201005-142345F_20201005-142457D

```

pg-primary ⇒ Attempt recovery from the specified backup

```

$ sudo pg_ctlcluster 12 demo stop
$ sudo -u postgres pgbackrest --stanza=demo --delta \
    --set=20201005-142345F_20201005-142517I \
    --type=time "--target=2020-10-05 14:25:05.993548+00" --target-
action=promote restore
$ sudo pg_ctlcluster 12 demo start
$ sudo -u postgres psql -c "select * from important_table"
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/log/postgresql/postgresql-12-demo.log
[filtered 3 lines of output]
LOG:  listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"
LOG:  database system was interrupted; last known up at 2020-10-05
14:25:17 UTC
LOG:  starting point-in-time recovery to 2020-10-05 14:25:05.993548+00
LOG:  restored log file "00000005.history" from archive
LOG:  restored log file "00000005000000000000000018" from archive
LOG:  redo starts at 0/18000028
LOG:  consistent recovery state reached at 0/18000100
LOG:  database system is ready to accept read only connections
LOG:  redo done at 0/18000100
[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 pg_ctlcluster 12 demo stop
$ sudo -u postgres pgbackrest --stanza=demo --delta \
    --type=time "--target=2020-10-05 14:25:05.993548+00" \
    --target-action=promote restore
$ sudo pg_ctlcluster 12 demo start
$ sudo -u postgres psql -c "select * from important_table"

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

Now the 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/log/postgresql/postgresql-12-demo.log
[filtered 5 lines of output]
LOG:  restored log file "00000005.history" from archive
LOG:  restored log file "00000006.history" from archive
LOG:  starting point-in-time recovery to 2020-10-05 14:25:05.993548+00
LOG:  restored log file "00000006.history" from archive
LOG:  restored log file "0000000400000000000000000016" from archive
[filtered 4 lines of output]
LOG:  database system is ready to accept read only connections
LOG:  restored log file "0000000500000000000000000017" from archive
LOG:  recovery stopping before commit of transaction 496, time
2020-10-05 14:25:16.499473+00
LOG:  redo done at 0/17022408
LOG:  last completed transaction was at log time 2020-10-05
14:25:04.492513+00
LOG:  selected new timeline ID: 7
LOG:  archive recovery complete
[filtered 2 lines of output]
```

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/postgresql/12/demo

[global]
process-max=4
repol-azure-account=pgbackrest
repol-azure-container=demo-container
repol-azure-key=YXpLZXk=
repol-cipher-
pass=zWaf6XtpjIVZC5444yXB+cgFDF17MxGlgkZSaoPvTGirhPygu4jOKOXf9L04vjfO
repol-cipher-type=aes-256-cbc
repol-path=/demo-repo
repol-retention-diff=2
repol-retention-full=2
repol-type=azure
start-fast=y

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

Shared access signatures may be used by setting the `repol-azure-key-type` option to `sas` and the `repol-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

P00 INFO: stanza-create command begin 2.30: --log-level-console=info
--log-level-stderr=off --no-log-timestamp --pg1-path=/var
/lib/postgresql/12/demo --repol-azure-account= --repol-azure-
container=demo-container --repol-azure-host=blob.core.windows.net
--repol-azure-key= --repol-cipher-pass= --repol-cipher-type=aes-256-cbc
--repol-path=/demo-repo --repol-type=azure --stanza=demo
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 \
--log-level-console=info backup

P00 INFO: backup command begin 2.30: --log-level-console=info --log-
level-stderr=off --no-log-timestamp --pg1-path=/var/lib/postgresql
/12/demo --process-max=4 --repol-azure-account= --repol-azure-
container=demo-container --repol-azure-host=blob.core.windows.net
--repol-azure-key= --repol-cipher-pass= --repol-cipher-type=aes-256-cbc
--repol-path=/demo-repo --repol-retention-diff=2 --repol-retention-
full=2 --repol-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 = 00000007000000000000000018, lsn =
0/18000028
[filtered 1279 lines of output]
P03 INFO: backup file /var/lib/postgresql/12/demo/base/1/13201 (0B,
100%)
P04 INFO: backup file /var/lib/postgresql/12/demo/base/1/13196 (0B,
```

```

100%)
P00    INFO: full backup size = 31MB
P00    INFO: execute non-exclusive pg_stop_backup() and wait for all WAL
segments to archive
P00    INFO: backup stop archive = 00000007000000000000000018, lsn =
0/18000138
      [filtered 4 lines of output]

```

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/postgresql/12/demo

[global]
process-max=4
repol-azure-account=pgbackrest
repol-azure-container=demo-container
repol-azure-key=YXpLZXk=
repol-cipher-
pass=zWaf6XtpjIVZC5444yXB+cgFDF17MxGlgkZSaoPvTGirhPygu4jOKOXf9L04vjfO
repol-cipher-type=aes-256-cbc
repol-path=/demo-repo
repol-retention-diff=2
repol-retention-full=2
repol-s3-bucket=demo-bucket
repol-s3-endpoint=s3.us-east-1.amazonaws.com
repol-s3-key=accessKey1
repol-s3-key-secret=verySecretKey1
repol-s3-region=us-east-1
repol-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 `repol-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.

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

```
P00 INFO: stanza-create command begin 2.30: --log-level-console=info
--log-level-stderr=off --no-log-timestamp --pg1-path=/var
/lib/postgresql/12/demo --repol-cipher-pass= --repol-cipher-type=aes-
256-cbc --repol-path=/demo-repo --repol-s3-bucket=demo-bucket --repol-
s3-endpoint=s3.us-east-1.amazonaws.com --repol-s3-key= --repol-s3-key-
secret= --repol-s3-region=us-east-1 --repol-type=s3 --stanza=demo
```

```
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 \
    --log-level-console=info backup

P00    INFO: backup command begin 2.30: --log-level-console=info --log-
level-stderr=off --no-log-timestamp --pgl-path=/var/lib/postgresql
/12/demo --process-max=4 --repol-cipher-pass= --repol-cipher-type=aes-
256-cbc --repol-path=/demo-repo --repol-retention-diff=2 --repol-
retention-full=2 --repol-s3-bucket=demo-bucket --repol-s3-
endpoint=s3.us-east-1.amazonaws.com --repol-s3-key= --repol-s3-key-
secret= --repol-s3-region=us-east-1 --repol-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 = 0000000700000000000000019, lsn =
0/19000028
    [filtered 1279 lines of output]
P03    INFO: backup file /var/lib/postgresql/12/demo/base/1/13201 (0B,
100%)
P04    INFO: backup file /var/lib/postgresql/12/demo/base/1/13196 (0B,
100%)

P00    INFO: full backup size = 31MB

P00    INFO: execute non-exclusive pg_stop_backup() and wait for all WAL
segments to archive
P00    INFO: backup stop archive = 0000000700000000000000019, lsn =
0/19000138
    [filtered 4 lines of output]
```

14 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 repository host.
- Run the `stanza-delete` command on the repository host.

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

pg-primary ⇒ Stop PostgreSQL cluster to be removed

```
$ sudo pg_ctlcluster 12 demo stop
```

pg-primary ⇒ Stop pgBackRest for the stanza

```
$ sudo -u postgres pgbackrest --stanza=demo --log-level-console=info stop
P00 INFO: stop command begin 2.30: --log-level-console=info --log-
level-stderr=off --no-log-timestamp --repol-cipher-pass= --repol-
cipher-type=aes-256-cbc --repol-path=/demo-repo --repol-s3-bucket=demo-
bucket --repol-s3-endpoint=s3.us-east-1.amazonaws.com --repol-s3-key=
--repol-s3-key-secret= --repol-s3-region=us-east-1 --repol-type=s3
--stanza=demo
P00 INFO: stop command end: completed successfully
```

pg-primary ⇒ Delete the stanza

```
$ sudo -u postgres pgbackrest --stanza=demo --log-level-console=info
stanza-delete
P00 INFO: stanza-delete command begin 2.30: --log-level-console=info
--log-level-stderr=off --no-log-timestamp --pg1-path=/var
/lib/postgresql/12/demo --repol-cipher-pass= --repol-cipher-type=aes-
256-cbc --repol-path=/demo-repo --repol-s3-bucket=demo-bucket --repol-
s3-endpoint=s3.us-east-1.amazonaws.com --repol-s3-key= --repol-s3-key-
secret= --repol-s3-region=us-east-1 --repol-type=s3 --stanza=demo
P00 INFO: stanza-delete command end: completed successfully
```

15 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.

15.1 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 adduser --disabled-password --gecos "" pgbackrest
```

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

build ⇒ Install dependencies

```
$ sudo apt-get install postgresql-client libxml2
```

repository ⇒ Copy pgBackRest binary from build host

```
$ sudo scp build:/build/pgbackrest-release-2.30/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
```

15.2 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/postgresql/.ssh
$ sudo -u postgres ssh-keygen -f /var/lib/postgresql/.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/postgresql/.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/postgresql/.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.*

15.3 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/postgresql/12/demo

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

The database host must be configured with the repository host/user. The default for the repol-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
repol-host/repol-host-user

```
[demo]
pg1-path=/var/lib/postgresql/12/demo

[global]
log-level-file=detail
repol-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
```

15.4 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
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.

15.5 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 pg_ctlcluster 12 demo stop
$ sudo -u postgres pgbackrest --stanza=demo --delta restore
$ sudo pg_ctlcluster 12 demo start
```

16 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/postgresql/12/demo

[global]
process-max=3
repol-path=/var/lib/pgbackrest
repol-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

stanza: demo
  status: ok
  cipher: none

  db (current)
    wal archive min/max (12-1):
00000008000000000000000000000001F/00000008000000000000000000000020

    full backup: 20201005-142706F
      timestamp start/stop: 2020-10-05 14:27:06 / 2020-10-05
14:27:25
      wal start/stop: 00000008000000000000000000000001F /
00000008000000000000000000000001F
      database size: 31MB, backup size: 31MB
      repository size: 3.7MB, repository backup size: 3.7MB

    full backup: 20201005-142727F
      timestamp start/stop: 2020-10-05 14:27:27 / 2020-10-05
14:27:35
      wal start/stop: 000000080000000000000000000000020 /
000000080000000000000000000000020
      database size: 31MB, backup size: 31MB
      repository size: 3.7MB, repository backup size: 3.7MB
```

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.

17 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
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
```

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
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
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
```

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
```

18 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.

18.1 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.

build ⇒ Install dependencies

```
$ sudo apt-get install postgresql-client libxml2
```

pg-standby ⇒ Copy pgBackRest binary from build host

```
$ sudo scp build:/build/pgbackrest-release-2.30/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
```

18.2 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/postgresql/.ssh
$ sudo -u postgres ssh-keygen -f /var/lib/postgresql/.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/postgresql/.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/postgresql/.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
```

18.3 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/postgresql/12/demo

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

The demo cluster must be created (even though it will be overwritten on restore) in order to create the PostgreSQL configuration files.

pg-standby ⇒ Create demo cluster

```
$ sudo pg_createcluster 12 demo
```

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 --delta --type=standby
restore
$ sudo -u postgres cat /var/lib/postgresql/12/demo/postgresql.auto.conf
# Do not edit this file manually!
# It will be overwritten by the ALTER SYSTEM command.

# Recovery settings generated by pgBackRest restore on 2020-10-05
14:23:05
restore_command = 'pgbackrest --stanza=demo archive-get %f "%p"'

# Recovery settings generated by pgBackRest restore on 2020-10-05
14:24:23
restore_command = 'pgbackrest --stanza=demo archive-get %f "%p"'

# Recovery settings generated by pgBackRest restore on 2020-10-05
14:24:47
restore_command = 'pgbackrest --stanza=demo archive-get %f "%p"'

# Recovery settings generated by pgBackRest restore on 2020-10-05
14:25:29
restore_command = 'pgbackrest --stanza=demo archive-get %f "%p"'
# Removed by pgBackRest restore on 2020-10-05 14:26:59 #
recovery_target_time = '2020-10-05 14:25:05.993548+00'
# Removed by pgBackRest restore on 2020-10-05 14:26:59 #
recovery_target_action = 'promote'

# Recovery settings generated by pgBackRest restore on 2020-10-05
14:26:59
restore_command = 'pgbackrest --stanza=demo archive-get %f "%p"'

# Recovery settings generated by pgBackRest restore on 2020-10-05
14:28:11
restore_command = 'pgbackrest --stanza=demo archive-get %f "%p"'
```

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:/etc/postgresql/12/demo/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 pg_ctlcluster 12 demo start
```

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/log/postgresql/postgresql-12-demo.log
```

```

[filtered 4 lines of output]
LOG:  listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"
LOG:  database system was interrupted; last known up at 2020-10-05
14:27:27 UTC
LOG:  entering standby mode
LOG:  restored log file "000000008.history" from archive
LOG:  restored log file "0000000080000000000000000020" from archive
LOG:  redo starts at 0/20000028
LOG:  consistent recovery state reached at 0/20000138
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";
      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;"
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()";
 pg_switch_wal | current_timestamp
-----+-----
0/21021750    | 2020-10-05 14:28:20.683628+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"
        message      |      current_timestamp
-----+-----
Important Data | 2020-10-05 14:28:26.110601+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
P00 INFO: check command begin 2.30: --log-level-console=info --log-
level-file=detail --log-level-stderr=off --no-log-timestamp --pgl-
path=/var/lib/postgresql/12/demo --repol-host=repository --stanza=demo
P00 INFO: switch wal not performed because this is a standby
P00 INFO: check command end: completed successfully

```

18.4 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";
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" \
    >> /etc/postgresql/12/demo/pg_hba.conf'
$ sudo pg_ctlcluster 12 demo reload

```

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]
pgl-path=/var/lib/postgresql/12/demo
recovery-option=primary_conninfo=host=172.17.0.5      port=5432
user=replicator

[global]
log-level-file=detail

```

```
repol-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/postgresql/.pgpass'
$ sudo -u postgres chmod 600 /var/lib/postgresql/.pgpass
```

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

pg-standby ⇒ Stop PostgreSQL and restore the demo standby cluster

```
$ sudo pg_ctlcluster 12 demo stop
$ sudo -u postgres pgbackrest --stanza=demo --delta --type=standby
restore
$ sudo -u postgres cat /var/lib/postgresql/12/demo/postgresql.auto.conf
# Do not edit this file manually!
# It will be overwritten by the ALTER SYSTEM command.

# Recovery settings generated by pgBackRest restore on 2020-10-05
14:23:05
restore_command = 'pgbackrest --stanza=demo archive-get %f "%p"'

# Recovery settings generated by pgBackRest restore on 2020-10-05
14:24:23
restore_command = 'pgbackrest --stanza=demo archive-get %f "%p"'

# Recovery settings generated by pgBackRest restore on 2020-10-05
14:24:47
restore_command = 'pgbackrest --stanza=demo archive-get %f "%p"'

# Recovery settings generated by pgBackRest restore on 2020-10-05
14:25:29
restore_command = 'pgbackrest --stanza=demo archive-get %f "%p"'
# Removed by pgBackRest restore on 2020-10-05 14:26:59 #
recovery_target_time = '2020-10-05 14:25:05.993548+00'
# Removed by pgBackRest restore on 2020-10-05 14:26:59 #
recovery_target_action = 'promote'

# Recovery settings generated by pgBackRest restore on 2020-10-05
14:26:59
restore_command = 'pgbackrest --stanza=demo archive-get %f "%p"'

# Recovery settings generated by pgBackRest restore on 2020-10-05
14:28:33
primary_conninfo = 'host=172.17.0.5 port=5432 user=replicator'
restore_command = 'pgbackrest --stanza=demo archive-get %f "%p"'
```

NOTE: The `primary_conninfo` setting has been written into the `postgresql.auto.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 `postgresql.auto.conf` file in place if that behavior is preferred.

pg-standby ⇒ Start PostgreSQL

```
$ sudo pg_ctlcluster 12 demo start
```


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/log/postgresql/postgresql-12-demo.log
[filtered 11 lines of output]
LOG:  database system is ready to accept read only connections
LOG:  restored log file "000000080000000000000000021" from archive
LOG:  started streaming WAL from primary at 0/22000000 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";
```

message	current_timestamp
Important Data	2020-10-05 14:28:41.406513+00

(1 row)

pg-standby ⇒ Query table on the standby

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

message	current_timestamp
Important Data	2020-10-05 14:28:41.918799+00

(1 row)

19 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/postgresql/12/demo

[global]
archive-async=y
log-level-file=detail
repol-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/postgresql/12/demo
recovery-option=primary_conninfo=host=172.17.0.5          port=5432
user=replicator

[global]
archive-async=y
log-level-file=detail
repol-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'"
ALTER ROLE
```

pg-standby ⇒ Restart standby to break connection

```
$ sudo pg_ctlcluster 12 demo restart
```

19.1 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
P00 INFO: check command begin 2.30: --log-level-console=info --log-
level-file=detail --log-level-stderr=off --no-log-timestamp --pgl-
path=/var/lib/postgresql/12/demo --repol-host=repository --stanza=demo
P00 INFO: WAL segment 0000000800000000000000027 successfully archived
to '/var/lib/pgbackrest/archive/demo/12-1/0000000800000000
/0000000800000000000000027-ebaad7538886bb429282523d91b168c4991ccb36.gz'
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
```

```

-----PROCESS START-----
P00 INFO: archive-push:async command begin 2.30: [/var/lib/postgresql
/12/demo/pg_wal] --archive-async --log-level-console=off --log-level-
file=detail --log-level-stderr=off --no-log-timestamp --pgl-path=/var
/lib/postgresql/12/demo --process-max=2 --repol-host=repository
--spool-path=/var/spool/pgbackrest --stanza=demo
P00 INFO: push 1 WAL file(s) to archive: 00000008000000000000000022
P01 DETAIL: pushed WAL file '00000008000000000000000022' to the archive
P00 INFO: archive-push:async command end: completed successfully

-----PROCESS START-----
P00 INFO: archive-push:async command begin 2.30: [/var/lib/postgresql
/12/demo/pg_wal] --archive-async --log-level-console=off --log-level-
file=detail --log-level-stderr=off --no-log-timestamp --pgl-path=/var
/lib/postgresql/12/demo --process-max=2 --repol-host=repository
--spool-path=/var/spool/pgbackrest --stanza=demo
P00 INFO: push 4 WAL file(s) to archive:
00000008000000000000000023...00000008000000000000000026
P01 DETAIL: pushed WAL file '00000008000000000000000023' to the archive
P02 DETAIL: pushed WAL file '00000008000000000000000024' to the archive
P01 DETAIL: pushed WAL file '00000008000000000000000025' to the archive
P02 DETAIL: pushed WAL file '00000008000000000000000026' to the archive
P00 INFO: archive-push:async command end: completed successfully

-----PROCESS START-----
P00 INFO: archive-push:async command begin 2.30: [/var/lib/postgresql
/12/demo/pg_wal] --archive-async --log-level-console=off --log-level-
file=detail --log-level-stderr=off --no-log-timestamp --pgl-path=/var
/lib/postgresql/12/demo --process-max=2 --repol-host=repository
--spool-path=/var/spool/pgbackrest --stanza=demo
P00 INFO: push 1 WAL file(s) to archive: 00000008000000000000000027
P01 DETAIL: pushed WAL file '00000008000000000000000027' to the archive
P00 INFO: archive-push:async command end: completed successfully

```

19.2 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
```

```

-----PROCESS START-----
P00 INFO: archive-get:async command begin 2.30:
r00000008000000000000000020. 00000008000000000000000021.

```

```

000000080000000000000000022, 000000080000000000000000023,
000000080000000000000000024, 000000080000000000000000025,
000000080000000000000000026, 000000080000000000000000027] --archive-async
--log-level-console=off --log-level-file=detail --log-level-stderr=off
--no-log-timestamp --pg1-path=/var/lib/postgresql/12/demo --process-
max=2 --repol-host=repository --spool-path=/var/spool/pgbackrest
--stanza=demo
P00 INFO: get 8 WAL file(s) from archive:
000000080000000000000000020...0000000800000000000000027

P01 DETAIL: found 000000080000000000000000020 in the archive
P02 DETAIL: found 000000080000000000000000021 in the archive

P01 DETAIL: unable to find 000000080000000000000000022 in the archive
P02 DETAIL: unable to find 000000080000000000000000023 in the archive
[filtered 20 lines of output]
P00 INFO: archive-get:async command begin 2.30:
[000000080000000000000000022, 000000080000000000000000023,
000000080000000000000000024, 000000080000000000000000025,
000000080000000000000000026, 000000080000000000000000027,
000000080000000000000000028, 000000080000000000000000029] --archive-async
--log-level-console=off --log-level-file=detail --log-level-stderr=off
--no-log-timestamp --pg1-path=/var/lib/postgresql/12/demo --process-
max=2 --repol-host=repository --spool-path=/var/spool/pgbackrest
--stanza=demo
P00 INFO: get 8 WAL file(s) from archive:
000000080000000000000000022...0000000800000000000000029

P01 DETAIL: found 000000080000000000000000022 in the archive
P02 DETAIL: found 000000080000000000000000023 in the archive
P02 DETAIL: found 000000080000000000000000025 in the archive
P01 DETAIL: found 000000080000000000000000024 in the archive

P01 DETAIL: unable to find 000000080000000000000000027 in the archive
P01 DETAIL: unable to find 000000080000000000000000028 in the archive
P01 DETAIL: unable to find 000000080000000000000000029 in the archive
P02 DETAIL: found 000000080000000000000000026 in the archive
P00 INFO: archive-get:async command end: completed successfully

[filtered 8 lines of output]
P02 DETAIL: unable to find 00000008000000000000000002D in the archive
P02 DETAIL: unable to find 00000008000000000000000002E in the archive
P01 DETAIL: found 000000080000000000000000027 in the archive
P00 INFO: archive-get:async command end: completed successfully

```

pg-primary ⇒ Fix streaming replication by changing the replication password

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

20 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
pg2-host/pg2-host-user and pg2-path
```

⇒

Configure

```
[demo]
pg1-host=pg-primary
pg1-path=/var/lib/postgresql/12/demo
pg2-host=pg-standby
pg2-path=/var/lib/postgresql/12/demo

[global]
backup-standby=y
process-max=3
repol-path=/var/lib/pgbackrest
repol-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
```

```
[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 = 0000000800000000000000029, lsn =
0/29000028

P00 INFO: wait for replay on the standby to reach 0/29000028
P00 INFO: replay on the standby reached 0/29000028
P01 INFO: backup file pg-primary:/var/lib/postgresql/12/demo/global
/pg_control (8KB, 0%) checksum 0aa92bfa26968248cb4daff8209e6d2b59ce0310

P04 INFO: backup file pg-standby:/var/lib/postgresql/12/demo
/base/13359/2674 (344KB, 15%) checksum
e9857616d26be7bb81dd8e4dccc4f31af2932789

P01 INFO: backup file pg-primary:/var/lib/postgresql/12/demo
/pg_logical/replorigin_checkpoint (8B, 15%) checksum
347fc8f2df71bd4436e38bd1516ccd7ea0d46532

P02 INFO: backup file pg-standby:/var/lib/postgresql/12/demo
/base/13359/2608 (456KB, 34%) checksum
6b714e16f30c51ecb88f59edcf431be791d41025
P03 INFO: backup file pg-standby:/var/lib/postgresql/12/demo
/base/13359/1249 (440KB, 53%) checksum
a14f8bafc904ad22f4916ed36dca398ff44e550c
[filtered 1292 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.

21 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` run on the repository host. 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 pg_ctlcluster 12 demo stop
```

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

pg-standby ⇒ Stop old cluster

```
$ sudo pg_ctlcluster 12 demo stop
```

Create the new cluster and perform upgrade.

pg-primary ⇒ Create new cluster and perform the upgrade

```
$ sudo -u postgres /usr/lib/postgresql/13/bin/initdb \
-D /var/lib/postgresql/13/demo -k -A peer
$ sudo pg_createcluster 13 demo
$ sudo -u postgres sh -c 'cd /var/lib/postgresql && \
/usr/lib/postgresql/13/bin/pg_upgrade \
--old-bindir=/usr/lib/postgresql/12/bin \
--new-bindir=/usr/lib/postgresql/13/bin \
--old-datadir=/var/lib/postgresql/12/demo \
--new-datadir=/var/lib/postgresql/13/demo \
--old-options=" -c config_file=/etc/postgresql/12/demo
/postgresql.conf" \
--new-options=" -c config_file=/etc/postgresql/13/demo
/postgresql.conf"'
```

```
[filtered 62 lines of output]
```

```
Creating script to delete old cluster
```

```
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:/etc/postgresql/13/demo/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/postgresql/13/demo

[global]
archive-async=y
log-level-file=detail
repol-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/postgresql/13/demo
recovery-option=primary_conninfo=host=172.17.0.5      port=5432
user=replicator

[global]
archive-async=y
log-level-file=detail
repol-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/postgresql/13/demo
pg2-host=pg-standby
pg2-path=/var/lib/postgresql/13/demo

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

pg-primary ⇒ Copy hba configuration

```
$ sudo cp /etc/postgresql/12/demo/pg_hba.conf \
    /etc/postgresql/13/demo/pg_hba.conf
```

Before starting the new cluster, the stanza-upgrade command must be run on the server

where the pgBackRest repository is located.

repository ⇒ Upgrade the stanza

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

P00    INFO: stanza-upgrade command begin 2.30: --no-backup-standby
--log-level-console=info --log-level-stderr=off --no-log-timestamp
--no-online --pg1-host=pg-primary --pg2-host=pg-standby --pg1-path=/var
/lib/postgresql/13/demo --pg2-path=/var/lib/postgresql/13/demo --repol-
path=/var/lib/pgbackrest --stanza=demo
P00    INFO: stanza-upgrade command end: completed successfully
```

Start the new cluster and confirm it is successfully installed.

pg-primary ⇒ Start new cluster

```
$ sudo pg_ctlcluster 13 demo start
```

Test configuration using the check command.

pg-primary ⇒ Check configuration

```
$ sudo -u postgres pg_lsclusters

Ver Cluster Port Status Owner    Data directory          Log file
12  demo    5432 down   postgres /var/lib/postgresql/12/demo /var/log
/postgresql/postgresql-12-demo.log
13  demo    5432 online postgres /var/lib/postgresql/13/demo /var/log
/postgresql/postgresql-13-demo.log

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

Remove the old cluster.

pg-primary ⇒ Remove old cluster

```
$ sudo pg_dropcluster 12 demo
```

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

pg-standby ⇒ Remove old cluster and create the new cluster

```
$ sudo pg_dropcluster 12 demo
$ sudo pg_createcluster 13 demo
```

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

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 --delta --type=standby  
restore
```

`pg-standby:/etc/postgresql/13/demo/postgresql.conf` ⇒ Configure PostgreSQL

```
hot_standby = on
```

`pg-standby` ⇒ Start PostgreSQL and check the pgBackRest configuration

```
$ sudo pg_ctlcluster 13 demo start  
$ 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/postgresql/13/demo  
pg2-host=pg-standby  
pg2-path=/var/lib/postgresql/13/demo  
  
[global]  
backup-standby=y  
process-max=3  
repo1-path=/var/lib/pgbackrest  
repo1-retention-full=2  
start-fast=y
```