

pgBackRest User Guide

Open Source PostgreSQL Backup and Restore Utility

Version 2.47



Crunchy Data Solutions, Inc.

September 13, 2023

Table of Contents

1	INTRODUCTION	2
2	CONCEPTS	3
2.1	BACKUP	3
2.2	RESTORE	3
2.3	WRITE AHEAD LOG (WAL)	3
2.4	ENCRYPTION	4
3	UPGRADING PGBACKREST	5
3.1	UPGRADING PGBACKREST FROM V1 TO V2	5
3.2	UPGRADING PGBACKREST FROM V2.X TO V2.Y	5
4	BUILD	6
5	INSTALLATION	7
6	QUICK START	9
6.1	SETUP DEMO CLUSTER	9
6.2	CONFIGURE CLUSTER STANZA	9
6.3	CREATE THE REPOSITORY	11
6.4	CONFIGURE ARCHIVING	12
6.5	CONFIGURE RETENTION	12
6.6	CONFIGURE REPOSITORY ENCRYPTION	13
6.7	CREATE THE STANZA	13
6.8	CHECK THE CONFIGURATION	14
6.9	PERFORM A BACKUP	15
6.10	SCHEDULE A BACKUP	16
6.11	BACKUP INFORMATION	17
6.12	RESTORE A BACKUP	18
7	MONITORING	20
7.1	IN POSTGRESQL	20
7.2	USING jq	21
8	BACKUP	23
8.1	FILE BUNDLING	23
8.2	BLOCK INCREMENTAL	24
8.3	BACKUP ANNOTATIONS	24
9	RETENTION	27
9.1	FULL BACKUP RETENTION	27
9.2	DIFFERENTIAL BACKUP RETENTION	28
9.3	ARCHIVE RETENTION	30
10	RESTORE	32

10.1 FILE OWNERSHIP	32
10.2 DELTA OPTION	32
10.3 RESTORE SELECTED DATABASES	33
11 POINT-IN-TIME RECOVERY	37
12 DELETE A STANZA	43
13 MULTIPLE REPOSITORIES	44
14 AZURE-COMPATIBLE OBJECT STORE SUPPORT	45
15 S3-COMPATIBLE OBJECT STORE SUPPORT	47
16 SFTP SUPPORT	51
17 GCS-COMPATIBLE OBJECT STORE SUPPORT	54
18 DEDICATED REPOSITORY HOST	56
18.1 INSTALLATION	56
18.2 SETUP PASSWORDLESS SSH	57
18.3 CONFIGURATION	58
18.4 CREATE AND CHECK STANZA	59
18.5 PERFORM A BACKUP	59
18.6 RESTORE A BACKUP	59
19 PARALLEL BACKUP / RESTORE	60
20 STARTING AND STOPPING	62
21 REPLICATION	64
21.1 INSTALLATION	64
21.2 SETUP PASSWORDLESS SSH	64
21.3 HOT STANDBY	65
21.4 STREAMING REPLICATION	69
22 ASYNCHRONOUS ARCHIVING	73
22.1 ARCHIVE PUSH	74
22.2 ARCHIVE GET	76
23 BACKUP FROM A STANDBY	78
24 UPGRADING POSTGRESQL	80

1 Introduction

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

Although the examples in this guide are targeted at `Debian/Ubuntu` and `PostgreSQL 12`, it should be fairly easy to apply the examples 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 of the executable may vary. While `pgBackRest` strives to operate consistently across versions of `PostgreSQL`, there are subtle differences between versions of `PostgreSQL` that may show up in this guide when illustrating certain examples, e.g. `PostgreSQL` path/file names and settings.

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

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

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

2 Concepts

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

2.1 Backup

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

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

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

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

2.2 Restore

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

2.3 Write Ahead Log (WAL)

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

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

2.4 Encryption

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

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

3 Upgrading pgBackRest

3.1 Upgrading pgBackRest from v1 to v2

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

However, there are a few caveats:

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

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

PostgreSQL and repository options must be indexed when using the new names introduced in v2, e.g. `pg1-host`, `pg1-path`, `repol-path`, `repol-type`, etc.

3.2 Upgrading pgBackRest from v2.x to v2.y

Upgrading from v2.x to v2.y is straight-forward. The repository format has not changed, so for most installations it is simply a matter of installing binaries for the new version. It is also possible to downgrade if you have not used new features that are unsupported by the older version.

4 Build

Installing pgBackRest from a package is preferable to building from source. See INSTALLATION for more information about packages.

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 2.47 of pgBackRest to /build path*

```
mkdir -p /build
wget -q -O - \
  https://github.com/pgbackrest/pgbackrest/archive/release/2.47.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 libyaml-dev libssh2-1-dev
```

build — *Configure and compile pgBackRest*

```
cd /build/pgbackrest-release-2.47/src && ./configure && make
```


5 Installation

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

Installing pgBackRest from a package is preferable to building from source. When installing from a package the rest of the instructions in this section are generally not required, but it is possible that a package will skip creating one of the directories or apply incorrect permissions. In that case it may be necessary to manually create directories or update permissions.

Debian/Ubuntu packages for pgBackRest are available at APT.POSTGRESQL.ORG.

If packages are not provided for your distribution/version you can BUILD FROM SOURCE and then install manually as shown here.

pg-primary — *Install dependencies*

```
sudo apt-get install postgresql-client libxml2 libssh2-1
```

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

```
sudo scp build:/build/pgbackrest-release-2.47/src/pgbackrest /usr/bin
sudo chmod 755 /usr/bin/pgbackrest
```

pgBackRest requires log and configuration directories and a configuration file.

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

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

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

pg-primary — *Make sure the installation worked*

```
sudo -u postgres pgbackrest
```

Output:

```
pgBackRest 2.47 - General help
```

Usage:

```
pgbackrest [options] [command]
```

Commands:

```
annotate    Add or modify backup annotation.
```

```
archive-get Get a WAL segment from the archive.
archive-push Push a WAL segment to the archive.
backup      Backup a database cluster.
check      Check the configuration.
expire     Expire backups that exceed retention.
help       Get help.
info       Retrieve information about backups.
repo-get   Get a file from a repository.
repo-ls    List files in a repository.
restore    Restore a database cluster.
server     pgBackRest server.
server-ping Ping pgBackRest server.
stanza-create Create the required stanza data.
stanza-delete Delete a stanza.
stanza-upgrade Upgrade a stanza.
start      Allow pgBackRest processes to run.
stop      Stop pgBackRest processes from running.
verify     Verify contents of the repository.
version    Get version.
```

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
```

Output:

```
Configuring already existing cluster (configuration: /etc/postgresql/12/demo, data  
: /var/lib/postgresql/12/demo, owner: 102:103)  
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
```

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 PostgreSQL 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'
```

Output:

```
pgBackRest 2.47 - '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]
repol-path=/var/lib/pgbackrest
```

Multiple repositories may also be configured. See [MULTIPLE REPOSITORIES](#) for details.

6.4 Configure Archiving

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

pg-primary:/etc/postgresql/12/demo/postgresql.conf — *Configure archive settings*

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

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

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

pg-primary — *Restart the demo cluster*

```
sudo 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+cgFDF17MxGlgkZSaoPvTGirhPygu4jOKOXf9LO4vjfO
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 to initialize the stanza. It is recommended that the `check` command be run after `stanza-create` to ensure archiving and backups are properly configured.

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

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

Output:

```
P00 INFO: stanza-create command begin 2.47: --exec-id=1292-04363209 --log-level-console=info --log-level-stderr=off --no-log-timestamp --pg1-path=/var/lib/postgresql/12/demo --repol-cipher-pass=<redacted> --repol-cipher-type=aes-256-cbc --repol-path=/var/lib/pgbackrest --stanza=demo
P00 INFO: stanza-create for stanza 'demo' on repol
P00 INFO: stanza-create command end: completed successfully
```

6.8 Check the Configuration

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

Note that `pg_create_restore_point('pgBackRest Archive Check')` and `pg_switch_xlog()/pg_switch_wal()` are called to force PostgreSQL to archive a WAL segment.

pg-primary — *Check the configuration*

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

Output:

```
P00 INFO: check command begin 2.47: --exec-id=1300-77fe66ac --log-level-console=info --log-level-stderr=off --no-log-timestamp --pg1-path=/var/lib/postgresql/12/demo --repol-cipher-pass=<redacted> --repol-cipher-type=aes-256-cbc --repol-path=/var/lib/pgbackrest --stanza=demo
P00 INFO: check repol configuration (primary)
P00 INFO: check repol archive for WAL (primary)
P00 INFO: WAL segment 00000001000000000000000000000001 successfully archived to '/var/lib/pgbackrest/archive/demo/12-1/0000000100000000/000000010000000000000001-d3149d83ae4878e937b5be7a91e8c011e260d386.gz' on repol
P00 INFO: check command end: completed successfully
```


6.9 Perform a Backup

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

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

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

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

pg-primary — *Backup the demo cluster*

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

Output:

```
P00 INFO: backup command begin 2.47: --exec-id=1327-bd30bed7 --log-level-console=
info --log-level-stderr=off --no-log-timestamp --pg1-path=/var/lib/postgresql
/12/demo --repol-cipher-pass=<redacted> --repol-cipher-type=aes-256-cbc --repol
-path=/var/lib/pgbackrest --repol-retention-full=2 --stanza=demo --start-fast
P00 WARN: no prior backup exists, incr backup has been changed to full
P00 INFO: execute non-exclusive backup start: backup begins after the requested
immediate checkpoint completes
P00 INFO: backup start archive = 00000001000000000000000002, lsn = 0/2000028
[filtered 3 lines of output]
P00 INFO: check archive for segment(s)
000000010000000000000002:000000010000000000000003
P00 INFO: new backup label = 20230724-061226F
P00 INFO: full backup size = 23.5MB, file total = 976
P00 INFO: backup command end: completed successfully
P00 INFO: expire command begin 2.47: --exec-id=1327-bd30bed7 --log-level-console=
info --log-level-stderr=off --no-log-timestamp --repol-cipher-pass=<redacted>
```

```
--repol-cipher-type=aes-256-cbc --repol-path=/var/lib/pgbackrest --repol-
retention-full=2 --stanza=demo
```

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

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

pg-primary — Differential backup of the demo cluster

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

Output:

```
[filtered 7 lines of output]
P00 INFO: check archive for segment(s)
      00000001000000000000000004:00000001000000000000000005
P00 INFO: new backup label = 20230724-061226F_20230724-061230D
P00 INFO: diff backup size = 8.3KB, file total = 976
P00 INFO: backup command end: completed successfully
P00 INFO: expire command begin 2.47: --exec-id=1354-c6c9003c --log-level-console=
info --log-level-stderr=off --no-log-timestamp --repol-cipher-pass=<redacted>
--repol-cipher-type=aes-256-cbc --repol-path=/var/lib/pgbackrest --repol-
retention-full=2 --stanza=demo
```

This time there was no warning because a full backup already existed. While incremental backups can be based on a full *or* differential backup, differential backups must be based on a full backup. A full backup can be performed by running the `backup` command with `--type=full`.

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

6.10 Schedule a Backup

Backups can be scheduled with utilities such as cron.

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

crontab:

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

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

6.11 Backup Information

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

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

```
sudo -u postgres pgbackrest info
```

Output:

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

db (current)
  wal archive min/max (12): 00000001000000000000000001/000000010000000000000005

  full backup: 20230724-061226F
    timestamp start/stop: 2023-07-24 06:12:26+00 / 2023-07-24 06:12:29+00
    wal start/stop: 00000001000000000000000002 / 000000010000000000000003
    database size: 23.5MB, database backup size: 23.5MB
    repol: backup set size: 2.8MB, backup size: 2.8MB

  diff backup: 20230724-061226F_20230724-061230D
    timestamp start/stop: 2023-07-24 06:12:30+00 / 2023-07-24 06:12:32+00
    wal start/stop: 00000001000000000000000004 / 000000010000000000000005
    database size: 23.5MB, database backup size: 8.3KB
    repol: backup set size: 2.8MB, backup size: 480B
    backup reference list: 20230724-061226F
```

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

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

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

in the archive and, in the case of multiple repositories, will be reported across all repositories unless the `--repo` option is set. Note that there may be gaps due to archive retention policies or other reasons.

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

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

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

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

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

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

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
```

Output:

```
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 Monitoring

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

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

7.1 In PostgreSQL

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

pg-primary — Load pgBackRest info function for PostgreSQL

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

Output:

```
-- An example of monitoring pgBackRest from within PostgreSQL  
--  
-- Use copy to export data from the pgBackRest info command into the jsonb  
-- type so it can be queried directly by PostgreSQL.  
  
-- Create monitor schema  
create schema monitor;  
  
-- Get pgBackRest info in JSON format  
create function monitor.pgbackrest_info()  
  returns jsonb AS $$  
declare  
  data jsonb;  
begin  
  -- Create a temp table to hold the JSON data  
  create temp table temp_pgbackrest_data (data text);  
  
  -- 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 replace(temp_pgbackrest_data.data, E'\n', '\n')::jsonb  
    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

```

Output:

```

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

```

Output:

```

 name | last_successful_backup | last_archived_wal
-----+-----+-----
"demo" | 2023-07-24 06:12:32+00 | 00000001000000000000000005
(1 row)

```

7.2 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'
```

Output:

```
1690179152
```

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'
```

Output:

```
"00000001000000000000000005"
```

■ **NOTE:** *This syntax requires `jq v1.5`.*

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

8 Backup

When multiple repositories are configured, pgBackRest will backup to the highest priority repository (e.g. `repo1`) unless the `--repo` option is specified.

pgBackRest does not have a built-in scheduler so it's best to run it from cron or some other scheduling mechanism.

See `PERFORM A BACKUP` for more details and examples.

8.1 File Bundling

Bundling files together in the repository saves time during the backup and some space in the repository. This is especially pronounced when the repository is stored on an object store such as S3. Per-file creation time on object stores is higher and very small files might cost as much to store as larger files.

The file bundling feature is enabled with the `repo-bundle` option.

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

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

[global]
repo1-bundle=y
repo1-cipher-pass=zWaf6XtpjIVZC5444yXB+cgFDF17MxGlgkZSaoPvTGirhPygu4jOKOXf9LO4vjfO
repo1-cipher-type=aes-256-cbc
repo1-path=/var/lib/pgbackrest
repo1-retention-full=2
start-fast=y

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

A full backup without file bundling will have 1000+ files in the backup path, but with bundling the total number of files is greatly reduced. An additional benefit is that zero-length files are not stored (except in the manifest), whereas in a normal backup each zero-length file is stored individually.

pg-primary — *Perform a full backup*

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

pg-primary — *Check file total*

```
sudo -u postgres find /var/lib/pgbackrest/backup/demo/latest/ -type f | wc -l
```

Output:

```
5
```

The `repo-bundle-size` and `repo-bundle-limit` options can be used for tuning, though the defaults should be optimal in most cases.

While file bundling is generally more efficient, the downside is that it is more difficult to manually retrieve files from the repository. It may not be ideal for deduplicated storage since each full backup will arrange files in the bundles differently. Lastly, file bundles cannot be resumed, so be careful not to set `repo-bundle-size` too high.

8.2 Block Incremental

Block incremental backups save space by only storing the parts of a file that have changed since the prior backup rather than storing the entire file.

The block incremental feature is enabled with the `repo-block` option and it works best when enabled for all backup types. File bundling must also be enabled.

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

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

[global]
repo1-block=y
repo1-bundle=y
repo1-cipher-pass=zWaf6XtpjIVZC5444yXB+cgFDF17MxGlgkZSaoPvTGirhPygu4jOKOXf9LO4vjfO
repo1-cipher-type=aes-256-cbc
repo1-path=/var/lib/pgbackrest
repo1-retention-full=2
start-fast=y

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

8.3 Backup Annotations

Users can attach informative key/value pairs to the backup. This option may be used multiple times to attach multiple annotations.

pg-primary — *Perform a full backup with annotations*

```
sudo -u postgres pgbackrest --stanza=demo --annotation=source="demo backup" \
  --annotation=key=value --type=full backup
```

Annotations are output by the `info` command text output when a backup is specified with `--set` and always appear in the JSON output.

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

```
sudo -u postgres pgbackrest --stanza=demo --set=20230724-061247F info
```

Output:

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

db (current)
  wal archive min/max (12): 00000002000000000000000007/000000020000000000000009

  full backup: 20230724-061247F
    timestamp start/stop: 2023-07-24 06:12:47+00 / 2023-07-24 06:12:48+00
    wal start/stop: 00000002000000000000000008 / 000000020000000000000009
    lsn start/stop: 0/8000028 / 0/9000050
    database size: 23.5MB, database backup size: 23.5MB
    rep1: backup size: 2.8MB
    database list: postgres (13427)
    annotation(s)
      key: value
      source: demo backup
```

Annotations included with the `backup` command can be added, modified, or removed afterwards using the `annotate` command.

pg-primary — *Change backup annotations*

```
sudo -u postgres pgbackrest --stanza=demo --set=20230724-061247F \
  --annotation=key= --annotation=new_key=new_value annotate
sudo -u postgres pgbackrest --stanza=demo --set=20230724-061247F info
```

Output:

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

db (current)
  wal archive min/max (12): 00000002000000000000000007/000000020000000000000009

  full backup: 20230724-061247F
    timestamp start/stop: 2023-07-24 06:12:47+00 / 2023-07-24 06:12:48+00
    wal start/stop: 00000002000000000000000008 / 000000020000000000000009
    lsn start/stop: 0/8000028 / 0/9000050
    database size: 23.5MB, database backup size: 23.5MB
    rep1: backup size: 2.8MB
    database list: postgres (13427)
    annotation(s)
```

```
new_key: new_value  
source: demo backup
```

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.

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

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

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-block=y
repol-bundle=y
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
```

Output:

```
[filtered 988 lines of output]
P00 INFO: repol: remove expired backup 20230724-061243F
P00 DETAIL: repol: 12-1 archive retention on backup 20230724-061247F, start =
  00000002000000000000000008
P00 INFO: repol: 12-1 remove archive, start = 000000020000000000000007, stop =
  000000020000000000000007
P00 INFO: expire command end: completed successfully
```

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

pg-primary — Perform a full backup

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

Output:

```
[filtered 11 lines of output]
P00 INFO: repol: expire full backup 20230724-061247F
P00 INFO: repol: remove expired backup 20230724-061247F
P00 INFO: repol: 12-1 remove archive, start = 000000020000000000000008, stop =
  00000002000000000000000A
P00 INFO: expire command end: completed successfully
```

The `20230724-061226F` full backup is expired and archive retention is based on the `20230724-061250F` 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-block=y
repol-bundle=y
repol-cipher-pass=zWaf6XtpjIVZC5444yXB+cgFDF17MxGlgkZSaoPvTGirhPygu4jOKOXf9L04vjfO
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
```

Output:

```
[filtered 10 lines of output]
P00 INFO: backup command end: completed successfully
P00 INFO: expire command begin 2.47: --exec-id=1838-a3748966 --log-level-console=
info --log-level-stderr=off --no-log-timestamp --repol-cipher-pass=<redacted>
--repol-cipher-type=aes-256-cbc --repol-path=/var/lib/pgbackrest --repol-
retention-diff=1 --repol-retention-full=2 --stanza=demo
P00 INFO: repol: expire diff backup set 20230724-061254F_20230724-061257D,
20230724-061254F_20230724-061259I
P00 INFO: repol: remove expired backup 20230724-061254F_20230724-061259I
P00 INFO: repol: remove expired backup 20230724-061254F_20230724-061257D
P00 INFO: expire command end: completed successfully
```

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-block=y
repol-bundle=y
repol-cipher-pass=zWaf6XtpjIVZC5444yXB+cgFDF17MxGlgkZSaoPvTGirhPygu4jOKOXf9LO4vjfO
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
```

Output:

```
[filtered 6 lines of output]
P00 INFO: backup stop archive = 0000000200000000000000017, lsn = 0/17000050
P00 INFO: check archive for segment(s)
0000000200000000000000016:0000000200000000000000017
P00 INFO: new backup label = 20230724-061254F_20230724-061305D
P00 INFO: diff backup size = 8.3KB, file total = 976
P00 INFO: backup command end: completed successfully
[filtered 2 lines of output]
```

pg-primary — *Expire archive*

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



```
--repol-retention-archive-type=diff --repol-retention-archive=1 expire
```

Output:

```
P00 INFO: expire command begin 2.47: --exec-id=1920-726e9d20 --log-level-console=
detail --log-level-stderr=off --no-log-timestamp --repol-cipher-pass=<redacted>
--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: repol: 12-1 archive retention on backup 20230724-061250F, start =
000000020000000000000000B, stop = 000000020000000000000000B
P00 DETAIL: repol: 12-1 archive retention on backup 20230724-061254F, start =
000000020000000000000000C, stop = 000000020000000000000000D
P00 DETAIL: repol: 12-1 archive retention on backup 20230724-061254F_20230724
-061302D, start = 0000000200000000000000012, stop = 000000020000000000000013
P00 DETAIL: repol: 12-1 archive retention on backup 20230724-061254F_20230724
-061305D, start = 0000000200000000000000016
P00 INFO: repol: 12-1 remove archive, start = 000000020000000000000000E, stop =
0000000200000000000000011
P00 INFO: repol: 12-1 remove archive, start = 0000000200000000000000014, stop =
0000000200000000000000015
P00 INFO: expire command end: completed successfully
```

The 20230724-061254F_20230724-061302D 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 20230724-061254F_20230724-061302D but before 20230724-061254F_20230724-061305D are removed. WAL segments generated after the new backup 20230724-061254F_20230724-061305D 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` command automatically defaults to selecting the latest backup from the first repository where backups exist (see QUICK START - RESTORE A BACKUP). The order in which the repositories are checked is dictated by the `pgbackrest.conf` (e.g. `repo1` will be checked before `repo2`). To select from a specific repository, the `--repo` option can be passed (e.g. `--repo=1`). The `--set` option can be passed if a backup other than the latest is desired.

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

Replication slots are not included per recommendation of PostgreSQL. See BACKING UP THE DATA DIRECTORY in the PostgreSQL documentation for more information.

The following sections introduce 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
```

Output:

```
[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 780 lines of output]
P01 DETAIL: restore file /var/lib/postgresql/12/demo/base/1/113 - exists and
matches backup (bundle 20230724-061254F/1/2552736, 8KB, 84.64%) checksum
e73a6fb61746a7a9459a81c2b83340e2923439f3
P01 DETAIL: restore file /var/lib/postgresql/12/demo/base/1/112 - exists and
matches backup (bundle 20230724-061254F/1/2552824, 8KB, 84.67%) checksum 23
fb025259d66e34784cd163d9e89852b68e6a38
P01 DETAIL: restore file /var/lib/postgresql/12/demo/PG_VERSION - exists and
matches backup (bundle 20230724-061254F/1/2552912, 3B, 84.67%) checksum
ad552e6dc057d1d825bf49df79d6b98eba846ebe
P01 DETAIL: restore file /var/lib/postgresql/12/demo/base/13427/2608_vm - exists
and matches backup (bundle 20230724-061254F/1/2552936, 8KB, 84.70%) checksum
3152243bd550b32137b8c8dc729355008cecb894
P01 DETAIL: restore file /var/lib/postgresql/12/demo/postgresql.auto.conf - exists
and matches backup (bundle 20230724-061254F/1/2553008, 229B, 84.70%) checksum
a129747bf0abc9c9ac72c603ce3b1531deae9c42
[filtered 238 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.

pg-primary — *Create two test databases*

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

Output:

```
CREATE DATABASE
```

```
sudo -u postgres psql -c "create database test2;"
```

Output:

```
CREATE DATABASE
```

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

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

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

Output:

```
CREATE TABLE
```

```
INSERT 0 1
```

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

Output:

```
CREATE TABLE
```

```
INSERT 0 1
```

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

pg-primary — *Perform a backup*

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

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

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

```
sudo -u postgres du -sh /var/lib/postgresql/12/demo/base/32768
```

Output:

```
7.8M /var/lib/postgresql/12/demo/base/32768
```

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=20230724-061254F_20230724-061314I info
```

Output:

```
[filtered 12 lines of output]
  repol: backup size: 1.9MB
  backup reference list: 20230724-061254F, 20230724-061254F_20230724-061305
  D
  database list: postgres (13427), test1 (32768), test2 (32769)
```

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

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

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

```
sudo pg_ctlcluster 12 demo stop
sudo -u postgres pgbackrest --stanza=demo --delta \
  --db-include=test2 --type=immediate --target-action=promote 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
```

Output:

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

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

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

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

Output:

```
psql: error: connection to server on socket "/var/run/postgresql/.s.PGSQL.5432"
failed: FATAL: relation mapping file "base/32768/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/32768
```

Output:

```
16K   /var/lib/postgresql/12/demo/base/32768
```

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

pg-primary — *Drop the test1 database*

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

Output:

```
DROP DATABASE
```

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

pg-primary — *List remaining databases*

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

Output:

```
oid | datname
-----+-----
  1 | template1
13426 | template0
13427 | postgres
32769 | 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 lsn, time, transaction id, or recovery point. For common recovery scenarios time-based recovery is arguably the most useful. A typical recovery scenario is to restore a table that was accidentally dropped or data that was accidentally deleted. Recovering a dropped table is more dramatic so that's the example given here but deleted data would be recovered in exactly the same way.

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

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

Output:

```
[filtered 4 lines of output]
message
-----
Important Data
(1 row)
```

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

pg-primary — *Get the time from PostgreSQL*

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

Output:

```
2023-07-24 06:13:27.043391+00
```

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

pg-primary — *Drop the important table*

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

```
select * from important_table;"
```

Output:

```
BEGIN
DROP TABLE
COMMITERROR: 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 2023-07-24 06:13:27.043391+00, and display postgresql.auto.conf*

```
sudo pg_ctlcluster 12 demo stop
sudo -u postgres pgbackrest --stanza=demo --delta \
  --type=time "--target=2023-07-24 06:13:27.043391+00" \
  --target-action=promote restore
sudo -u postgres cat /var/lib/postgresql/12/demo/postgresql.auto.conf
```

Output:

```
[filtered 11 lines of output]
# Removed by pgBackRest restore on 2023-07-24 06:13:29 # recovery_target = '
  immediate'
# Removed by pgBackRest restore on 2023-07-24 06:13:29 # recovery_target_action =
  'promote'
# Removed by pgBackRest restore on 2023-07-24 06:13:29 # recovery_target_timeline
  = 'current'

# Recovery settings generated by pgBackRest restore on 2023-07-24 06:13:29
restore_command = 'pgbackrest --stanza=demo archive-get %f "%p"'
recovery_target_time = '2023-07-24 06:13:27.043391+00'
recovery_target_action = 'promote'
```

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

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

```
sudo pg_ctlcluster 12 demo start
sudo -u postgres psql -c "select * from important_table"
```

Output:

```
message
-----
```


Important Data
(1 row)

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

pg-primary — *Examine the PostgreSQL log output*

```
sudo -u postgres cat /var/log/postgresql/postgresql-12-demo.log
```

Output:

```
[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 2023-07-24 06:13:24 UTC
LOG: starting point-in-time recovery to 2023-07-24 06:13:27.043391+00
LOG: restored log file "00000004.history" from archive
LOG: restored log file "000000040000000000000001A" from archive
[filtered 2 lines of output]
LOG: database system is ready to accept read only connections
LOG: restored log file "000000040000000000000001B" from archive
LOG: recovery stopping before commit of transaction 496, time 2023-07-24
    06:13:28.351049+00
LOG: redo done at 0/1B019E88
LOG: last completed transaction was at log time 2023-07-24 06:13:25.751754+00
LOG: selected new timeline ID: 5
LOG: archive recovery complete
[filtered 2 lines of output]
```

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

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

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

Output:

```
BEGIN
DROP TABLE
COMMITERROR: relation "important_table" does not exist
LINE 1: ...le important_table; commit; select * from important_...
        ^
```

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

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

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

Output:

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

db (current)
  wal archive min/max (12): 000000020000000000000000B/000000050000000000000001C

  full backup: 20230724-061250F
    timestamp start/stop: 2023-07-24 06:12:50+00 / 2023-07-24 06:12:53+00
    wal start/stop: 000000020000000000000000B / 000000020000000000000000B
    database size: 23.5MB, database backup size: 23.5MB
    repl: backup size: 2.8MB

  full backup: 20230724-061254F
    timestamp start/stop: 2023-07-24 06:12:54+00 / 2023-07-24 06:12:55+00
    wal start/stop: 000000020000000000000000C / 000000020000000000000000D
    database size: 23.5MB, database backup size: 23.5MB
    repl: backup size: 2.8MB

  diff backup: 20230724-061254F_20230724-061305D
    timestamp start/stop: 2023-07-24 06:13:05+00 / 2023-07-24 06:13:07+00
    wal start/stop: 0000000200000000000000016 / 0000000200000000000000017
    database size: 23.5MB, database backup size: 8.3KB
    repl: backup size: 464B
    backup reference list: 20230724-061254F

  incr backup: 20230724-061254F_20230724-061314I
    timestamp start/stop: 2023-07-24 06:13:14+00 / 2023-07-24 06:13:16+00
    wal start/stop: 0000000300000000000000019 / 0000000300000000000000019
    database size: 38.8MB, database backup size: 15.8MB
    repl: backup size: 1.9MB
    backup reference list: 20230724-061254F, 20230724-061254F_20230724-061305
      D

  diff backup: 20230724-061254F_20230724-061324D
    timestamp start/stop: 2023-07-24 06:13:24+00 / 2023-07-24 06:13:25+00
    wal start/stop: 000000040000000000000001A / 000000040000000000000001A
    database size: 31.1MB, database backup size: 8.2MB
    repl: backup size: 970.9KB
    backup reference list: 20230724-061254F

  incr backup: 20230724-061254F_20230724-061335I
```

```
timestamp start/stop: 2023-07-24 06:13:35+00 / 2023-07-24 06:13:36+00
wal start/stop: 000000050000000000000001C / 000000050000000000000001C
database size: 31.1MB, database backup size: 2.2MB
repol: backup size: 25.0KB
backup reference list: 20230724-061254F, 20230724-061254F_20230724-061324
D
```

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

```
sudo pg_ctlcluster 12 demo stop
sudo -u postgres pgbackrest --stanza=demo --delta \
  --set=20230724-061254F_20230724-061335I \
  --type=time "--target=2023-07-24 06:13:27.043391+00" --target-action=promote
  restore
sudo pg_ctlcluster 12 demo start
sudo -u postgres psql -c "select * from important_table"
```

Output:

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

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

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

```
sudo -u postgres cat /var/log/postgresql/postgresql-12-demo.log
```

Output:

```
[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 2023-07-24 06:13:35 UTC
LOG: starting point-in-time recovery to 2023-07-24 06:13:27.043391+00
LOG: restored log file "00000005.history" from archive
LOG: restored log file "000000050000000000000001C" from archive
LOG: redo starts at 0/1C000028
LOG: consistent recovery state reached at 0/1C000100
LOG: database system is ready to accept read only connections
LOG: redo done at 0/1C000100
[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=2023-07-24 06:13:27.043391+00" \
  --target-action=promote restore
sudo pg_ctlcluster 12 demo start
sudo -u postgres psql -c "select * from important_table"

```

Output:

```

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

```

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

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

```

sudo -u postgres cat /var/log/postgresql/postgresql-12-demo.log

```

Output:

```

      [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 2023-07-24 06:13:27.043391+00
LOG: restored log file "00000006.history" from archive
LOG: restored log file "000000040000000000000001A" from archive
      [filtered 4 lines of output]
LOG: database system is ready to accept read only connections
LOG: restored log file "000000050000000000000001B" from archive
LOG: recovery stopping before commit of transaction 497, time 2023-07-24
      06:13:34.391759+00
LOG: redo done at 0/1B022530
LOG: last completed transaction was at log time 2023-07-24 06:13:25.751754+00
LOG: selected new timeline ID: 7
LOG: archive recovery complete
LOG: database system is ready to accept connections

```

12 Delete a Stanza

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

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

To delete a stanza:

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

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

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

pg-primary — Stop PostgreSQL cluster to be removed

```
sudo pg_ctlcluster 12 demo stop
```

pg-primary — Stop pgBackRest for the stanza

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

Output:

```
P00 INFO: stop command begin 2.47: --exec-id=2641-632739a1 --log-level-console=
info --log-level-stderr=off --no-log-timestamp --stanza=demo
P00 INFO: stop command end: completed successfully
```

pg-primary — Delete the stanza from one repository

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

Output:

```
P00 INFO: stanza-delete command begin 2.47: --exec-id=2650-0ed27a9f --log-level-
console=info --log-level-stderr=off --no-log-timestamp --pg1-path=/var/lib/
postgresql/12/demo --repo=1 --repol-cipher-pass=<redacted> --repol-cipher-type=
aes-256-cbc --repol-path=/var/lib/pgbackrest --stanza=demo
P00 INFO: stanza-delete command end: completed successfully
```

13 Multiple Repositories

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

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

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

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

14 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.

WARNING: Do not enable “*hierarchical namespace*” as this will cause errors during expire.

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

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

[global]
process-max=4
repo1-block=y
repo1-bundle=y
repo1-cipher-pass=zWaf6XtpjIVZC5444yXB+cgFDF17MxGlgkZSaoPvTGirhPygu4jOKOXf9L04vjf0
repo1-cipher-type=aes-256-cbc
repo1-path=/var/lib/pgbackrest
repo1-retention-diff=2
repo1-retention-full=2
repo2-azure-account=pgbackrest
repo2-azure-container=demo-container
repo2-azure-key=YXpLZXk=
repo2-path=/demo-repo
repo2-retention-full=4
repo2-type=azure
start-fast=y

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

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

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

pg-primary — *Create the stanza*

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

Output:

```
P00 INFO: stanza-create command begin 2.47: --exec-id=2723-30df1642 --log-level-console=info --log-level-stderr=off --no-log-timestamp --pg1-path=/var/lib/postgresql/12/demo --repo2-azure-account=<redacted> --repo2-azure-container=demo-container --repo2-azure-key=<redacted> --repo1-cipher-pass=<redacted> --
```

```

repol-cipher-type=aes-256-cbc --repol-path=/var/lib/pgbackrest --repo2-path=/
demo-repo --repo2-type=azure --stanza=demo
P00 INFO: stanza-create for stanza 'demo' on repo1
P00 INFO: stanza-create for stanza 'demo' on repo2
P00 INFO: stanza-create command end: completed successfully

```

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

pg-primary — *Backup the demo cluster*

```

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

```

Output:

```

P00 INFO: backup command begin 2.47: --exec-id=2732-853d3e2f --log-level-console=
info --log-level-stderr=off --no-log-timestamp --pgl-path=/var/lib/postgresql
/12/demo --process-max=4 --repo=2 --repo2-azure-account=<redacted> --repo2-
azure-container=demo-container --repo2-azure-key=<redacted> --repol-block --
repol-bundle --repol-cipher-pass=<redacted> --repol-cipher-type=aes-256-cbc --
repol-path=/var/lib/pgbackrest --repo2-path=/demo-repo --repol-retention-diff=2
--repol-retention-full=2 --repo2-retention-full=4 --repo2-type=azure --stanza=
demo --start-fast
P00 WARN: no prior backup exists, incr backup has been changed to full
P00 INFO: execute non-exclusive backup start: backup begins after the requested
immediate checkpoint completes
P00 INFO: backup start archive = 0000000700000000000000001D, lsn = 0/1D000028
[filtered 3 lines of output]
P00 INFO: check archive for segment(s) 0000000700000000000000001D
:0000000700000000000000001D
P00 INFO: new backup label = 20230724-061353F
P00 INFO: full backup size = 31.1MB, file total = 1282
P00 INFO: backup command end: completed successfully
P00 INFO: expire command begin 2.47: --exec-id=2732-853d3e2f --log-level-console=
info --log-level-stderr=off --no-log-timestamp --repo=2 --repo2-azure-account=<
redacted> --repo2-azure-container=demo-container --repo2-azure-key=<redacted>
--repol-cipher-pass=<redacted> --repol-cipher-type=aes-256-cbc --repol-path=/
var/lib/pgbackrest --repo2-path=/demo-repo --repol-retention-diff=2 --repol-
retention-full=2 --repo2-retention-full=4 --repo2-type=azure --stanza=demo

```


15 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
repo1-block=y
repo1-bundle=y
repo1-cipher-pass=zWaf6XtpjIVZC5444yXB+cgFDF17MxGlgkZSaoPvTGirhPygu4jOKOXf9LO4vjfO
repo1-cipher-type=aes-256-cbc
repo1-path=/var/lib/pgbackrest
repo1-retention-diff=2
repo1-retention-full=2
repo2-azure-account=pgbackrest
repo2-azure-container=demo-container
repo2-azure-key=YXpLZXk=
repo2-path=/demo-repo
repo2-retention-full=4
repo2-type=azure
repo3-path=/demo-repo
repo3-retention-full=4
repo3-s3-bucket=demo-bucket
repo3-s3-endpoint=s3.us-east-1.amazonaws.com
repo3-s3-key=accessKey1
repo3-s3-key-secret=verySecretKey1
repo3-s3-region=us-east-1
repo3-type=s3
start-fast=y

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

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

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

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

Sample Amazon S3 Policy:

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

```

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

```

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

pg-primary — Create the stanza

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

Output:

```

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

```

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

pg-primary — Backup the demo cluster

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

Output:

```

P00 INFO: backup command begin 2.47: --exec-id=2798-d4b5db56 --log-level-console=
info --log-level-stderr=off --no-log-timestamp --pg1-path=/var/lib/postgresql
/12/demo --process-max=4 --repo=3 --repo2-azure-account=<redacted> --repo2-
azure-container=demo-container --repo2-azure-key=<redacted> --repo1-block --
repo1-bundle --repo1-cipher-pass=<redacted> --repo1-cipher-type=aes-256-cbc --
repo1-path=/var/lib/pgbackrest --repo2-path=/demo-repo --repo3-path=/demo-repo
--repo1-retention-diff=2 --repo1-retention-full=2 --repo2-retention-full=4 --
repo3-retention-full=4 --repo3-s3-bucket=demo-bucket --repo3-s3-endpoint=s3.us-
east-1.amazonaws.com --repo3-s3-key=<redacted> --repo3-s3-key-secret=<redacted>
--repo3-s3-region=us-east-1 --repo2-type=azure --repo3-type=s3 --stanza=demo
--start-fast
P00 WARN: no prior backup exists, incr backup has been changed to full
P00 INFO: execute non-exclusive backup start: backup begins after the requested
immediate checkpoint completes
P00 INFO: backup start archive = 0000000700000000000000001E, lsn = 0/1E000028
    [filtered 3 lines of output]
P00 INFO: check archive for segment(s) 0000000700000000000000001E
:0000000700000000000000001F
P00 INFO: new backup label = 20230724-061402F

```

```
P00 INFO: full backup size = 31.1MB, file total = 1282
P00 INFO: backup command end: completed successfully
P00 INFO: expire command begin 2.47: --exec-id=2798-d4b5db56 --log-level-console=
info --log-level-stderr=off --no-log-timestamp --repo=3 --repo2-azure-account=<
redacted> --repo2-azure-container=demo-container --repo2-azure-key=<redacted>
--repo1-cipher-pass=<redacted> --repo1-cipher-type=aes-256-cbc --repo1-path=/
var/lib/pgbackrest --repo2-path=/demo-repo --repo3-path=/demo-repo --repo1-
retention-diff=2 --repo1-retention-full=2 --repo2-retention-full=4 --repo3-
retention-full=4 --repo3-s3-bucket=demo-bucket --repo3-s3-endpoint=s3.us-east
-1.amazonaws.com --repo3-s3-key=<redacted> --repo3-s3-key-secret=<redacted> --
repo3-s3-region=us-east-1 --repo2-type=azure --repo3-type=s3 --stanza=demo
```

16 SFTP Support

pgBackRest supports locating repositories on SFTP hosts. SFTP file transfer is relatively slow so commands benefit by increasing `process-max` to parallelize file transfer.

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

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

[global]
process-max=4
repo1-block=y
repo1-bundle=y
repo1-cipher-pass=zWaf6XtpjIVZC5444yXB+cgFDF17MxGlgkZSaoPvTGirhPygu4jOKOXf9LO4vjfO
repo1-cipher-type=aes-256-cbc
repo1-path=/var/lib/pgbackrest
repo1-retention-diff=2
repo1-retention-full=2
repo2-azure-account=pgbackrest
repo2-azure-container=demo-container
repo2-azure-key=YXpLZXk=
repo2-path=/demo-repo
repo2-retention-full=4
repo2-type=azure
repo3-path=/demo-repo
repo3-retention-full=4
repo3-s3-bucket=demo-bucket
repo3-s3-endpoint=s3.us-east-1.amazonaws.com
repo3-s3-key=accessKey1
repo3-s3-key-secret=verySecretKey1
repo3-s3-region=us-east-1
repo3-type=s3
repo4-bundle=y
repo4-path=/demo-repo
repo4-sftp-host=sftp-server
repo4-sftp-host-key-hash-type=sha1
repo4-sftp-host-user=pgbackrest
repo4-sftp-private-key-file=/var/lib/postgresql/.ssh/id_rsa_sftp
repo4-sftp-public-key-file=/var/lib/postgresql/.ssh/id_rsa_sftp.pub
repo4-type=sftp
start-fast=y

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

When utilizing SFTP, if `libssh2` is compiled against OpenSSH then `repo4-sftp-public-key-file` is optional.

pg-primary — *Generate SSH keypair for SFTP backup*

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

sftp-server — *Copy pg-primary SFTP backup public key to sftp-server*

```
sudo -u pgbackrest mkdir -m 750 -p /home/pgbackrest/.ssh
(sudo ssh root@pg-primary cat /var/lib/postgresql/.ssh/id_rsa_sftp.pub) | \
  sudo -u pgbackrest tee -a /home/pgbackrest/.ssh/authorized_keys
```

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

pg-primary — *Create the stanza*

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

Output:

```
[filtered 6 lines of output]
P00 INFO: stanza 'demo' already exists on repo3 and is valid
P00 INFO: stanza-create for stanza 'demo' on repo4
P00 INFO: stanza-create command end: completed successfully
```

pg-primary — *Backup the demo cluster*

```
sudo -u postgres pgbackrest --stanza=demo --repo=4 \
  --log-level-console=info backup
```

Output:

```
P00 INFO: backup command begin 2.47: --exec-id=2872-a740dc16 --log-level-console=
info --log-level-stderr=off --no-log-timestamp --pg1-path=/var/lib/postgresql
/12/demo --process-max=4 --repo=4 --repo2-azure-account=<redacted> --repo2-
azure-container=demo-container --repo2-azure-key=<redacted> --repo1-block --
repo1-bundle --repo4-bundle --repo1-cipher-pass=<redacted> --repo1-cipher-type=
aes-256-cbc --repo1-path=/var/lib/pgbackrest --repo2-path=/demo-repo --repo3-
path=/demo-repo --repo4-path=/demo-repo --repo1-retention-diff=2 --repo1-
retention-full=2 --repo2-retention-full=4 --repo3-retention-full=4 --repo3-s3-
bucket=demo-bucket --repo3-s3-endpoint=s3.us-east-1.amazonaws.com --repo3-s3-
key=<redacted> --repo3-s3-key-secret=<redacted> --repo3-s3-region=us-east-1 --
repo4-sftp-host=sftp-server --repo4-sftp-host-key-hash-type=sha1 --repo4-sftp-
host-user=pgbackrest --repo4-sftp-private-key-file=/var/lib/postgresql/.ssh/
id_rsa_sftp --repo4-sftp-public-key-file=/var/lib/postgresql/.ssh/id_rsa_sftp.
pub --repo2-type=azure --repo3-type=s3 --repo4-type=sftp --stanza=demo --start-
fast
P00 WARN: option 'repo4-retention-full' is not set for 'repo4-retention-full-type=
count', the repository may run out of space
      HINT: to retain full backups indefinitely (without warning), set option '
      repo4-retention-full' to the maximum.
```

```
P00 WARN: no prior backup exists, incr backup has been changed to full
P00 INFO: execute non-exclusive backup start: backup begins after the requested
        immediate checkpoint completes
P00 INFO: backup start archive = 00000007000000000000000020, lsn = 0/20000028
        [filtered 3 lines of output]
P00 INFO: check archive for segment(s)
        00000007000000000000000020:00000007000000000000000021
P00 INFO: new backup label = 20230724-061408F
P00 INFO: full backup size = 31.1MB, file total = 1282
P00 INFO: backup command end: completed successfully
P00 INFO: expire command begin 2.47: --exec-id=2872-a740dc16 --log-level-console=
        info --log-level-stderr=off --no-log-timestamp --repo=4 --repo2-azure-account=<
        redacted> --repo2-azure-container=demo-container --repo2-azure-key=<redacted>
        --repo1-cipher-pass=<redacted> --repo1-cipher-type=aes-256-cbc --repo1-path=/
        var/lib/pgbackrest --repo2-path=/demo-repo --repo3-path=/demo-repo --repo4-path
        =/demo-repo --repo1-retention-diff=2 --repo1-retention-full=2 --repo2-retention
        -full=4 --repo3-retention-full=4 --repo3-s3-bucket=demo-bucket --repo3-s3-
        endpoint=s3.us-east-1.amazonaws.com --repo3-s3-key=<redacted> --repo3-s3-key-
        secret=<redacted> --repo3-s3-region=us-east-1 --repo4-sftp-host=sftp-server --
        repo4-sftp-host-key-hash-type=sha1 --repo4-sftp-host-user=pgbackrest --repo4-
        sftp-private-key-file=/var/lib/postgresql/.ssh/id_rsa_sftp --repo4-sftp-public-
        key-file=/var/lib/postgresql/.ssh/id_rsa_sftp.pub --repo2-type=azure --repo3-
        type=s3 --repo4-type=sftp --stanza=demo
P00 INFO: expire command end: completed successfully
```

17 GCS-Compatible Object Store Support

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

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

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

[global]
process-max=4
repo1-block=y
repo1-bundle=y
repo1-cipher-pass=zWaf6XtpjIVZC5444yXB+cgFDF17MxGlgkZSaoPvTGirhPygu4jOKOXf9L04vjfO
repo1-cipher-type=aes-256-cbc
repo1-path=/var/lib/pgbackrest
repo1-retention-diff=2
repo1-retention-full=2
repo2-azure-account=pgbackrest
repo2-azure-container=demo-container
repo2-azure-key=YXpLZXk=
repo2-path=/demo-repo
repo2-retention-full=4
repo2-type=azure
repo3-path=/demo-repo
repo3-retention-full=4
repo3-s3-bucket=demo-bucket
repo3-s3-endpoint=s3.us-east-1.amazonaws.com
repo3-s3-key=accessKey1
repo3-s3-key-secret=verySecretKey1
repo3-s3-region=us-east-1
repo3-type=s3
repo4-bundle=y
repo4-path=/demo-repo
repo4-sftp-host=sftp-server
repo4-sftp-host-key-hash-type=sha1
repo4-sftp-host-user=pgbackrest
repo4-sftp-private-key-file=/var/lib/postgresql/.ssh/id_rsa_sftp
repo4-sftp-public-key-file=/var/lib/postgresql/.ssh/id_rsa_sftp.pub
repo4-type=sftp
repo5-gcs-bucket=demo-bucket
repo5-gcs-key=/etc/pgbackrest/gcs-key.json
repo5-path=/demo-repo
repo5-type=gcs
start-fast=y
```



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

When running in GCE set `repo5-gcs-key-type=auto` to automatically authenticate using the instance service account.

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

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

18 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.

18.1 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
```

Installing `pgBackRest` from a package is preferable to building from source. When installing from a package the rest of the instructions in this section are generally not required, but it is possible that a package will skip creating one of the directories or apply incorrect permissions. In that case it may be necessary to manually create directories or update permissions.

Debian/Ubuntu packages for `pgBackRest` are available at APT.POSTGRESQL.ORG.

If packages are not provided for your distribution/version you can BUILD FROM SOURCE and then install manually as shown here.

repository — Install dependencies

```
sudo apt-get install postgresql-client libxml2 libssh2-1
```

repository — Copy `pgBackRest` binary from build host

```
sudo scp build:/build/pgbackrest-release-2.47/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

```

18.2 Setup Passwordless SSH

pgBackRest can use passwordless SSH to enable communication between the hosts. It is also possible to use TLS, see [SETUP TLS](#).

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.*

18.3 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
```

PostgreSQL configuration may be found in the **CONFIGURE ARCHIVING** section.

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.

18.4 Create and Check Stanza

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
```

18.5 Perform a Backup

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

repository — *Backup the demo cluster*

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

Output:

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

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

18.6 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
```

19 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
```

Output:

```
stanza: demo
  status: ok
  cipher: none

db (current)
  wal archive min/max (12): 00000008000000000000000027/00000008000000000000000029

  full backup: 20230724-061440F
    timestamp start/stop: 2023-07-24 06:14:40+00 / 2023-07-24 06:14:44+00
    wal start/stop: 00000008000000000000000027 / 00000008000000000000000027
```

```
database size: 31.1MB, database backup size: 31.1MB  
repol: backup set size: 3.7MB, backup size: 3.7MB
```

```
full backup: 20230724-061446F
```

```
timestamp start/stop: 2023-07-24 06:14:46+00 / 2023-07-24 06:14:50+00
```

```
wal start/stop: 0000000800000000000000028 / 0000000800000000000000029
```

```
database size: 31.1MB, database backup size: 31.1MB
```

```
repol: backup set size: 3.7MB, 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.

20 Starting and Stopping

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

pg-primary — *Stop the pgBackRest services*

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

New pgBackRest processes will no longer run.

repository — *Attempt a backup*

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

Output:

```
P00 WARN: unable to check pgl: [StopError] raised from remote-0 ssh protocol on '
pg-primary': stop file exists for all stanzas
P00 ERROR: [056]: unable to find primary cluster - cannot proceed
HINT: are all available clusters in recovery?
```

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

pg-primary — *Stop the pgBackRest services again*

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

Output:

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

Start pgBackRest processes again with the `start` command.

pg-primary — *Start the pgBackRest services*

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

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

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

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

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

repository — *Attempt a backup*

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


Output:

```
P00 WARN: unable to check pg1: [StopError] raised from remote-0 ssh protocol on '
  pg-primary': stop file exists for stanza demo
P00 ERROR: [056]: unable to find primary cluster - cannot proceed
      HINT: are all available clusters in recovery?
```

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
```

21 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.

21.1 Installation

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

Installing pgBackRest from a package is preferable to building from source. When installing from a package the rest of the instructions in this section are generally not required, but it is possible that a package will skip creating one of the directories or apply incorrect permissions. In that case it may be necessary to manually create directories or update permissions.

Debian/Ubuntu packages for pgBackRest are available at APT.POSTGRESQL.ORG.

If packages are not provided for your distribution/version you can BUILD FROM SOURCE and then install manually as shown here.

pg-standby — *Install dependencies*

```
sudo apt-get install postgresql-client libxml2 libssh2-1
```

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

```
sudo scp build:/build/pgbackrest-release-2.47/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
```

21.2 Setup Passwordless SSH

pgBackRest can use passwordless SSH to enable communication between the hosts. It is also possible to use TLS, see SETUP TLS.

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
```

21.3 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
```

Output:

```
# Do not edit this file manually!
# It will be overwritten by the ALTER SYSTEM command.

# Recovery settings generated by pgBackRest restore on 2023-07-24 06:12:34
restore_command = 'pgbackrest --stanza=demo archive-get %f "%p"'

# Recovery settings generated by pgBackRest restore on 2023-07-24 06:13:08
restore_command = 'pgbackrest --stanza=demo archive-get %f "%p"'

# Recovery settings generated by pgBackRest restore on 2023-07-24 06:13:18
restore_command = 'pgbackrest --stanza=demo archive-get %f "%p"'
# Removed by pgBackRest restore on 2023-07-24 06:13:43 # recovery_target = '
  immediate'
# Removed by pgBackRest restore on 2023-07-24 06:13:43 # recovery_target_action =
  'promote'
# Removed by pgBackRest restore on 2023-07-24 06:13:43 # recovery_target_timeline
  = 'current'

# Recovery settings generated by pgBackRest restore on 2023-07-24 06:13:43
restore_command = 'pgbackrest --stanza=demo archive-get %f "%p"'
# Removed by pgBackRest restore on 2023-07-24 06:14:34 # recovery_target_time =
  '2023-07-24 06:13:27.043391+00'
# Removed by pgBackRest restore on 2023-07-24 06:14:34 # recovery_target_action =
  'promote'

# Recovery settings generated by pgBackRest restore on 2023-07-24 06:14:34
restore_command = 'pgbackrest --stanza=demo archive-get %f "%p"'

# Recovery settings generated by pgBackRest restore on 2023-07-24 06:15:04
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
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
```

Output:

```
[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 2023-07-24 06:14:46 UTC
LOG: entering standby mode
LOG: restored log file "00000008.history" from archive
LOG: restored log file "00000008000000000000000028" from archive
LOG: redo starts at 0/28000028
LOG: restored log file "00000008000000000000000029" from archive
LOG: consistent recovery state reached at 0/29000050
LOG: database system is ready to accept read only connections
```

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

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

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

Output:

```
[filtered 4 lines of output]
message
-----
```

```
Important Data
(1 row)
```

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

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

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

Output:

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

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

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

pg-primary — *Call pg_switch_wal()*

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

Output:

```
pg_switch_wal | current_timestamp
-----+-----
0/2A021810 | 2023-07-24 06:15:11.45949+00
(1 row)
```

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

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

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

Output:

```
message | current_timestamp
-----+-----
Important Data | 2023-07-24 06:15:12.518+00
(1 row)
```

Check the standby configuration for access to the repository.

pg-standby — *Check the configuration*

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

Output:

```
P00 INFO: check command begin 2.47: --exec-id=1370-6f78dbc5 --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: check repol (standby)
P00 INFO: switch wal not performed because this is a standby
P00 INFO: check command end: completed successfully
```

21.4 Streaming Replication

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

Streaming replication requires a user with the replication privilege.

pg-primary — Create replication user

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

Output:

```
CREATE ROLE
```

The `pg_hba.conf` file must be updated to allow the standby to connect as the replication user. Be sure to replace the IP address below with the actual IP address of your **pg-standby**. 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 172.17.0.8/32 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=172.17.0.6 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 \
    "172.17.0.6:*: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
```

Output:

```
# Do not edit this file manually!
# It will be overwritten by the ALTER SYSTEM command.

# Recovery settings generated by pgBackRest restore on 2023-07-24 06:12:34
restore_command = 'pgbackrest --stanza=demo archive-get %f "%p"'

# Recovery settings generated by pgBackRest restore on 2023-07-24 06:13:08
restore_command = 'pgbackrest --stanza=demo archive-get %f "%p"'

# Recovery settings generated by pgBackRest restore on 2023-07-24 06:13:18
restore_command = 'pgbackrest --stanza=demo archive-get %f "%p"'
# Removed by pgBackRest restore on 2023-07-24 06:13:43 # recovery_target = '
    immediate'
# Removed by pgBackRest restore on 2023-07-24 06:13:43 # recovery_target_action =
    'promote'
# Removed by pgBackRest restore on 2023-07-24 06:13:43 # recovery_target_timeline
    = 'current'

# Recovery settings generated by pgBackRest restore on 2023-07-24 06:13:43
restore_command = 'pgbackrest --stanza=demo archive-get %f "%p"'
# Removed by pgBackRest restore on 2023-07-24 06:14:34 # recovery_target_time =
    '2023-07-24 06:13:27.043391+00'
# Removed by pgBackRest restore on 2023-07-24 06:14:34 # recovery_target_action =
    'promote'

# Recovery settings generated by pgBackRest restore on 2023-07-24 06:14:34
restore_command = 'pgbackrest --stanza=demo archive-get %f "%p"'
```



```
# Recovery settings generated by pgBackRest restore on 2023-07-24 06:15:15
primary_conninfo = 'host=172.17.0.6 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
```

Output:

```
[filtered 11 lines of output]
LOG: database system is ready to accept read only connections
LOG: restored log file "000000080000000000000002A" from archive
LOG: started streaming WAL from primary at 0/2B000000 on timeline 8
```

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

pg-primary — Create a new table on the primary

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

Output:

```
[filtered 4 lines of output]
message |      current_timestamp
-----+-----
Important Data | 2023-07-24 06:15:21.853479+00
(1 row)
```

pg-standby — Query table on the standby

```
sudo -u postgres psql -c " \
```

```
select *, current_timestamp from stream_table"
```

Output:

```
message | current_timestamp
-----+-----
Important Data | 2023-07-24 06:15:22.112147+00
(1 row)
```

22 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=172.17.0.6 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'"
```

Output:

```
ALTER ROLE
```

pg-standby — Restart standby to break connection

```
sudo pg_ctlcluster 12 demo restart
```

22.1 Archive Push

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

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

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

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

pg-primary — *Test parallel asynchronous archiving*

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

Output:

```
P00 INFO: check command begin 2.47: --exec-id=3461-3e54c2c3 --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: check repol configuration (primary)
P00 INFO: check repol archive for WAL (primary)
P00 INFO: WAL segment 00000008000000000000000030 successfully archived to '/var/lib/
pgbackrest/archive/demo/12-1/0000000800000000/000000080000000000000030-8
c60edb46769fd5920827b8852cb90c971bfab3b.gz' on repol
P00 INFO: check command end: completed successfully
```

Now the log file will contain parallel, asynchronous activity.

pg-primary — *Check results in the log*

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

Output:

```
-----PROCESS START-----
P00 INFO: archive-push:async command begin 2.47: [/var/lib/postgresql/12/demo/
pg_wal] --archive-async --exec-id=3446-d7fc3077 --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: 0000000800000000000000002B
P01 DETAIL: pushed WAL file '0000000800000000000000002B' to the archive
P00 INFO: archive-push:async command end: completed successfully

-----PROCESS START-----
P00 INFO: archive-push:async command begin 2.47: [/var/lib/postgresql/12/demo/
pg_wal] --archive-async --exec-id=3464-66117434 --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: 0000000800000000000000002C
...0000000800000000000000002F
P02 DETAIL: pushed WAL file '0000000800000000000000002D' to the archive
P01 DETAIL: pushed WAL file '0000000800000000000000002C' to the archive
```

```

P02 DETAIL: pushed WAL file '00000008000000000000000002E' to the archive
P01 DETAIL: pushed WAL file '00000008000000000000000002F' to the archive
P00 INFO: archive-push:async command end: completed successfully

-----PROCESS START-----
P00 INFO: archive-push:async command begin 2.47: [/var/lib/postgresql/12/demo/
pg_wal] --archive-async --exec-id=3480-2052eabf --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: push 1 WAL file(s) to archive: 000000080000000000000000030
P01 DETAIL: pushed WAL file '000000080000000000000000030' to the archive
P00 INFO: archive-push:async command end: completed successfully

```

22.2 Archive Get

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

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

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

pg-standby — *Check results in the log*

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

Output:

```

-----PROCESS START-----
P00 INFO: archive-get:async command begin 2.47: [000000080000000000000000028,
000000080000000000000000029, 00000008000000000000000002A, 000000080000000000000002B,
00000008000000000000000002C, 00000008000000000000000002D, 000000080000000000000002E,
00000008000000000000000002F] --archive-async --exec-id=1580-c904aca5 --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:
000000080000000000000000028...000000080000000000000002F
P02 DETAIL: found 000000080000000000000000029 in the rep1: 12-1 archive
P01 DETAIL: found 000000080000000000000000028 in the rep1: 12-1 archive
P02 DETAIL: found 00000008000000000000000002A in the rep1: 12-1 archive

```

```

P00 DETAIL: unable to find 000000080000000000000002B in the archive
P00 INFO: archive-get:async command end: completed successfully
      [filtered 14 lines of output]
P00 INFO: archive-get:async command begin 2.47: [000000080000000000000002B,
000000080000000000000002C, 000000080000000000000002D, 000000080000000000000002E,
000000080000000000000002F, 0000000800000000000000030, 0000000800000000000000031,
0000000800000000000000032] --archive-async --exec-id=1624-49d991b0 --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: 000000080000000000000002B
...0000000800000000000000032
P01 DETAIL: found 000000080000000000000002B in the repol: 12-1 archive
P02 DETAIL: found 000000080000000000000002C in the repol: 12-1 archive
P01 DETAIL: found 000000080000000000000002D in the repol: 12-1 archive
P02 DETAIL: found 000000080000000000000002E in the repol: 12-1 archive
P01 DETAIL: found 000000080000000000000002F in the repol: 12-1 archive
P02 DETAIL: found 0000000800000000000000030 in the repol: 12-1 archive
P00 DETAIL: unable to find 0000000800000000000000031 in the archive
P00 INFO: archive-get:async command end: completed successfully
      [filtered 10 lines of output]

```

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

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

Output:

```
ALTER ROLE
```

23 Backup from a Standby

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

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

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

Output:

```
[filtered 2 lines of output]
P00 INFO: execute non-exclusive backup start: backup begins after the requested
      immediate checkpoint completes
P00 INFO: backup start archive = 00000008000000000000000032, lsn = 0/32000028
P00 INFO: wait for replay on the standby to reach 0/32000028
P00 INFO: replay on the standby reached 0/32000028
P00 INFO: check archive for prior segment 00000008000000000000000031
P01 DETAIL: backup file pg-primary:/var/lib/postgresql/12/demo/global/pg_control
      (8KB, 0.34%) checksum 53bbd8a42492d666a4d9477118dc71d6c47fa8a5
P01 DETAIL: backup file pg-primary:/var/lib/postgresql/12/demo/pg_logical/
      replorigin_checkpoint (8B, 0.34%) checksum 347
      fc8f2df71bd4436e38bd1516ccd7ea0d46532
P02 DETAIL: backup file pg-standby:/var/lib/postgresql/12/demo/base/13427/2608
      (456KB, 19.86%) checksum 2e52d240454d156ce30e811ffec8544931b5cd63
P03 DETAIL: backup file pg-standby:/var/lib/postgresql/12/demo/base/13427/1249
      (440KB, 38.69%) checksum 0a1cbfa99b0dc14173f5032b413da9bed4fc2796
```


[filtered 1293 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.

24 Upgrading PostgreSQL

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

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

pg-primary — *Stop old cluster*

```
sudo 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/14/bin/initdb \
  -D /var/lib/postgresql/14/demo -k -A peer
sudo pg_createcluster 14 demo
sudo -u postgres sh -c 'cd /var/lib/postgresql && \
  /usr/lib/postgresql/14/bin/pg_upgrade \
  --old-bindir=/usr/lib/postgresql/12/bin \
  --new-bindir=/usr/lib/postgresql/14/bin \
  --old-datadir=/var/lib/postgresql/12/demo \
  --new-datadir=/var/lib/postgresql/14/demo \
  --old-options=" -c config_file=/etc/postgresql/12/demo/postgresql.conf" \
  --new-options=" -c config_file=/etc/postgresql/14/demo/postgresql.conf"'
```

Output:

```
[filtered 68 lines of output]
Checking for extension updates          ok

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

Configure the new cluster settings and port.

pg-primary:/etc/postgresql/14/demo/postgresql.conf — Configure PostgreSQL

```
archive_command = 'pgbackrest --stanza=demo archive-push %p'
archive_mode = on
max_wal_senders = 3
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/14/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/14/demo
recovery-option=primary_conninfo=host=172.17.0.6 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/14/demo
pg2-host=pg-standby
```

```
pg2-path=/var/lib/postgresql/14/demo

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

pg-primary — Copy hba configuration

```
sudo cp /etc/postgresql/12/demo/pg_hba.conf \
  /etc/postgresql/14/demo/pg_hba.conf
```

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

pg-primary — Upgrade the stanza

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

Output:

```
P00 INFO: stanza-upgrade command begin 2.47: --exec-id=3898-d868dc7a --log-level-
console=info --log-level-file=detail --log-level-stderr=off --no-log-timestamp
--no-online --pg1-path=/var/lib/postgresql/14/demo --repo1-host=repository --
stanza=demo
P00 INFO: stanza-upgrade for stanza 'demo' on repo1
P00 INFO: stanza-upgrade command end: completed successfully
```

Start the new cluster and confirm it is successfully installed.

pg-primary — Start new cluster

```
sudo pg_ctlcluster 14 demo start
```

Test configuration using the `check` command.

pg-primary — Check configuration

```
sudo -u postgres pg_lsclusters
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 14 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
```

Output:

```
P00 WARN: unable to check pg2: [DbConnectError] raised from remote-0 ssh protocol
on 'pg-standby': unable to connect to 'dbname='postgres' port=5432': connection
to server on socket "/var/run/postgresql/.s.PGSQL.5432" failed: No such file
or directory
    Is the server running locally and accepting connections on that socket
    ?
```

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/14/demo/postgresql.conf — *Configure PostgreSQL*

```
hot_standby = on
```

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

```
sudo pg_ctlcluster 14 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/14/demo
pg2-host=pg-standby
pg2-path=/var/lib/postgresql/14/demo
```

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