



Introduction to pgBackRest

Release 2.40.0

EnterpriseDB Corporation

Jan 05, 2023

CONTENTS

1	'Installation'	3
1.1	Available platforms	3
1.2	Debian/Ubuntu	3
1.3	RHEL/CentOS	3
1.4	EDB Postgres Advanced Server	4
2	Quick Start	5
2.1	Configuration	5
2.2	Backups	6
2.3	Backup Information	6
2.4	Restore	7
2.5	Recovery target	7
2.6	Glossary	8
2.7	pgBackRest	8
2.7.1	restore command	8
2.8	PostgreSQL	8
3	'Recommended Settings'	9
3.1	Encryption	9
3.2	Maximum Number of Processes	9
3.3	Backup Fast Start	9
3.4	Delta	10
3.5	Log Levels	10
3.6	Glossary	10
3.7	pgBackRest	10
4	Retention Policy	11
4.1	Full backup retention	11
4.2	Differential backup retention	11
4.3	Archive retention	12
4.4	repo1-retention-archive-type	12
4.5	repo1-retention-archive	12
4.6	Glossary	13
4.7	pgBackRest	13
5	Running pgBackRest Locally on the Database Host'	15
5.1	Description	15
5.2	Configuration	15
5.3	Global Section	15
5.3.1	Mount Points	15

5.3.2	S3-compatible Object Stores	16
5.3.3	Azure-compatible Object Stores	16
5.3.4	GCS-compatible Object Stores	16
5.3.5	Recommended Settings	17
5.3.6	Retention Policy	17
5.4	Stanza Section	17
5.5	Setup Archiving	18
5.6	Backup and Restore	18
5.7	Glossary	18
5.8	pgBackRest	18
5.9	PostgreSQL	19
6	Running pgBackRest from a Dedicated Repository Host'	21
6.1	Description	21
6.2	Create a Dedicated User on the Repository Host	21
6.3	Setup Passwordless SSH Connection	21
6.4	Configuration	22
6.5	Backup Server	22
6.6	Database server(s)	23
6.7	Setup Archiving	23
6.8	Backup and Restore	24
6.9	Glossary	24
6.10	pgBackRest	24
6.11	PostgreSQL	24
7	'Multiple Repositories'	25
7.1	Configuration	25
7.2	Initialization	26
7.3	Stanza Create Command	26
7.4	Check Command	26
7.5	Archive Push Command	26
7.6	Backups	28
7.7	Backup Command	28
7.8	Info Command	29
7.9	Restore	32
7.10	Restore Command	34
7.11	Archive Get Command	35
8	'TLS connections'	39
8.1	Description	39
8.2	Create a Dedicated User on the Repository Host	39
8.3	Certificates generation	40
8.4	Configuration	41
8.5	Perform a backup	44
8.6	Prepare the servers for Streaming Replication	45
8.7	Setup the standby servers	45
8.8	Take backups from the standby servers	46
8.9	Conclusion	47
9	'Appendix'	49
9.1	Concepts	49
9.2	Backup	49
9.3	Restore	49
9.4	Write Ahead Log (WAL)	50
9.5	Encryption	50

9.6	Configuration Reference	50
9.7	Global	50
9.8	Stanza	51
9.9	Command Line Arguments and Environment Variables	51
9.10	Options Precedence	51
9.11	Configuring Multiple Database Hosts in a Stanza	52
10	'Frequently Asked Questions'	53

pgBackRest is an easy-to-use backup and restore tool that aims to enable your organization to implement disaster recovery solutions for PostgreSQL databases.

Note: Looking for pgBackRest documentation? Head over to <http://www.pgbackrest.org>

pgBackRest [v2.40](#) is the current stable release. Release notes are on the [Releases](#) page.

EDB collaborates with the community on this open-source software. The packages are provided by the **PostgreSQL** community. All of the use cases shown in this document are fully tested and supported by **EDBPostgresAdvanced-Server** .

‘INSTALLATION’

1.1 Available platforms

pgBackRest packages are available on the following platforms.

1.2 Debian/Ubuntu

To install pgBackRest, configure the *PostgreSQL* **apt** repository and then run:

```
$ sudo apt-get install pgbackrest
```

The following additional packages will be installed:

- postgresql-client-common
- postgresql-common

1.3 RHEL/CentOS

pgBackRest can be installed from the *PostgreSQL* **yum** repository, and requires Extra Packages for Enterprise Linux ([EPEL](#)).

Once you configured the repositories, run the following command to install pgBackRest:

```
$ sudo yum install pgbackrest
```

The following additional packages may be installed to satisfy the dependencies:

- libpq5
- libzstd

1.4 EDB Postgres Advanced Server

The PGDG packages will fetch **PostgreSQL** libraries and create the `postgres` system user.

If **EDBPostgresAdvancedServer** is already installed on the system, the libraries installed at that time would be used.

By default, the packages create the following directories owned by `postgres` :

- `/var/lib/pgbackrest`
- `/var/log/pgbackrest`
- `/var/spool/pgbackrest`

The `pg_hba.conf` settings should allow the `postgres` user to execute the `pgbackrest` command.

If you are using **EDBPostgresAdvancedServer** , the `enterprisedb` system user will execute the `pgbackrest` command. The following commands will change the ownership of the `pgBackRest` directories:

```
$ sudo chown -R enterprisedb: /var/lib/pgbackrest
$ sudo chown -R enterprisedb: /var/log/pgbackrest
$ sudo chown -R enterprisedb: /var/spool/pgbackrest
```

QUICK START

This section walks you through basic pgBackRest configuration process and how to take a first backup of the database cluster running locally. It also gives a brief introduction of the `backup`, `restore`, and `info` commands.

The ‘Installation’ procedure must be followed before proceeding to this section.

2.1 Configuration

The default pgBackRest configuration file location is `/etc/pgbackrest/pgbackrest.conf`. If it does not exist, then `/etc/pgbackrest.conf` is used next.

For more information about configuration parameters, see the
appendix section.

The following example configures a `demo` stanza to take a backup of a database cluster running on the local host.

For **EDBPostgresAdvancedServer** :

```
[global]
repo1-path=/var/lib/edb/as13/backups

[demo]
pg1-path=/var/lib/edb/as13/data
pg1-user=enterprisedb
pg1-port=5444
```

For **PostgreSQL** :

```
[global]
repo1-path=/var/lib/pgsql/13/backups

[demo]
pg1-path=/var/lib/pgsql/13/data
pg1-user=postgres
pg1-port=5432
```

Once pgBackRest is configured, set up the database archiver process:

```
# postgresql.conf

archive_mode = on
archive_command = pgbackrest --stanza=demo archive-push %p
```

As changing the `archive_mode` parameter requires a service restart, and changing the `archive_command` only requires a configuration reload, we recommend enabling `archive_mode` with an empty `archive_command` (or pointing to `/bin/true`) when initiating a new database cluster.

Assume the identity of a system user (`postgres` or `enterprisedb`) and initiate the pgBackRest repository:

```
$ pgbackrest --stanza=demo --log-level-console=info stanza-create
```

Check the configuration and the archiving process:

```
$ pgbackrest --stanza=demo --log-level-console=info check
```

2.2 Backups

When invoking pgBackRest, you can specify any one of the backup type:

- Full backup - `full` - Copies all database cluster files. - No dependencies on previous backups.

For example:

```
$ pgbackrest --stanza=demo --type=full backup
```

- Incremental backup - `incr` - Copies only the database cluster files that have changed since the last backup. - Depends on other backups to restore the incremental backup.

For example:

```
$ pgbackrest --stanza=demo --type=incr backup
```

- Differential backup - `diff` - Copies only the database cluster files that have changed since the last full backup. - Similar to an incremental backup, but always depends on the last full backup.

For example:

```
$ pgbackrest --stanza=demo --type=diff backup
```

If there is no full backup to base an incremental or differential backup on, pgBackRest will run in full backup mode instead.

2.3 Backup Information

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

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

  db (current)
    wal archive min/max (13-1): 000000010000000000000001/000000010000000000000007

    full backup: 20201019-162148F
      timestamp start/stop: 2020-10-19 16:21:48 / 2020-10-19 16:21:58
```

(continues on next page)

(continued from previous page)

```
wal start/stop: 000000010000000000000003 / 000000010000000000000003
database size: 23.9MB, backup size: 23.9MB
repository size: 2.9MB, repository backup size: 2.9MB
...
```

By default, the `info` command provides a human-readable summary of backups for the stanza requested. Without specifying `--stanza`, the `info` command will operate on all stanzas.

2.4 Restore

Use the `restore` command to restore a backup. The `--delta` option allows to keep the existing database cluster files and only recover as required, where as without the `--delta` option all files will be restored. Before running the restore command, stop the database cluster and (unless the `--delta` option is used) remove all files from the data directory, and from tablespaces.

```
$ pgbackrest --stanza=demo restore
```

Currently, the only way to verify the consistency of a backup is to restore it. We then recommend to try restore backups on a regular basis.

2.5 Recovery target

By default, the `restore` command will restore the latest backup and apply all of the archived WAL files found in the repository.

To choose the recovery target type for the restore, include the `--type` command-line option with one of the following keywords:

- `default` - recover to the end of the archive stream.
- `immediate` - recovery only until the database becomes consistent.
- `lsn` - recover to the LSN (Log Sequence Number).
- `name` - recover to the restore point.
- `xid` - recover to the transaction ID.
- `time` - recover to a specific timestamp.
- `preserve` - preserve the existing recovery settings.
- `standby` - add `standby_mode=on` or `standby.signal`.
- `none` - no recovery settings provided.

When `--type` is set to `lsn`, `name`, transaction ID (`xid`), or `time`, use the `--target` option to specify a restore point.

pgBackRest will automatically select the correct backup set to restore if a recovery target time is specified. Otherwise, the latest backup set will be used. To restore a specific backup set, use the `--set` option.

To change the `recovery_target_action` PostgreSQL setting, use the restore `--target-action` option:

- `pause` - pause when recovery target is reached (default).
- `promote` - promote and switch timeline when recovery target is reached.
- `shutdown` - shutdown server when recovery target is reached.

2.6 Glossary

2.7 pgBackRest

Delta log-level-console pg-path pg-port pg-user repo-path

2.7.1 restore command

restore --set restore --target restore --target-action restore --type

2.8 PostgreSQL

archive_command archive_mode recovery_target_action standby_mode

‘RECOMMENDED SETTINGS’

This section walks you through the recommended settings while using pgBackRest.

3.1 Encryption

Use the following settings to encrypt the files in the repository `repo1` :

```
repo1-cipher-pass=FIXME  
repo1-cipher-type=aes-256-cbc
```

The `cipher-pass` should be **base64** encoded. For example, to generate a random passphrase use the following setting:

```
$ openssl rand -base64 48
```

3.2 Maximum Number of Processes

Use the following setting to set the maximum number of processes for compression usage and file transfer:

```
process-max=FIXME
```

Each process will perform compression and transfer to make the command run faster, but do not set `process-max` so high that it impacts server load and performance.

3.3 Backup Fast Start

Use the `start-fast=y` setting to force a checkpoint to start the backup quickly. This is achieved inside pgBackRest by passing **true** to the `fast` parameter of the `pg_start_backup()` database function. The backup will then begin immediately rather than waiting for the next regular checkpoint triggered by the database server itself.

3.4 Delta

Use the `delta=y` setting to restore or backup using checksums.

During a restore, the data and tablespaces directories are expected to be present, but empty. This option performs a delta restore using checksums.

During a backup, this option will use checksums instead of timestamps to determine if files will be copied.

3.5 Log Levels

The output of each command will be printed in the console and a log file.

Use the following `log-level-console` and `log-level-file` settings to apply the log level:

- `off` ;
- `error` ;
- `warn` ;
- `info` ;
- `detail` ;
- `debug` ;
- `trace` .

CAUTION: Trace-level logging may expose secrets such as keys and passwords.

To add more output in the log file in case the executed command encounters an error and to avoid re-executing the command for more information, use the following settings:

```
log-level-console=info
log-level-file=debug
```

3.6 Glossary

3.7 pgBackRest

Delta log-level-console log-level-file process-max repo-cipher-pass repo-cipher-type

RETENTION POLICY

Setting retention policy options allows to remove obsolete backups. Those settings will then be a trade-off between saving space and data retention.

4.1 Full backup retention

Full backup retention is determined by the `repo1-retention-full-type` and `repo1-retention-full` options.

The `repo1-retention-full-type` option can be set to `count` (default) to retain a specific number of full backups. As an example, if `repo1-retention-full-type=count` and `repo1-retention-full=2`, pgBackRest will retain the 2 last successful full backups.

Alternatively the `repo1-retention-full-type` option can be set to `time` to retain full backups for a specific time duration. With this configuration, at least one full backup older than the retention period will remain to fulfill the retention requirement. All other full backups older than the `repo1-retention-full` value, measured in days, will be removed from the repository. As an example, if `repo1-retention-full-type=time` and `repo1-retention-full=30` (days), one full backup older than 30 days will be preserved. If in this example the repository would hold five full backups - one 25 days old, one 35 days old, and the rest older - the 35-days-old full backup would still be required to restore to the period between 30 and 25 days ago, and as such is kept even though it is older than the configured retention period. All other older backups will be removed during retention maintenance.

When a full backup expires, all differential and incremental backups associated with it will also expire. When the `repo1-retention-full` option is not defined, a warning will be issued.

Archived WAL files that are older than the remaining oldest full backup will be automatically removed, but this behavior can be changed with the archive retention settings (`repo1-retention-archive-type` and `repo1-retention-archive`).

Set the `repo1-retention-full` option to its maximum value (9999999) to retain all backups indefinitely.

4.2 Differential backup retention

Differential backup retention is determined by the `repo1-retention-diff` option.

When a differential backup expires, all incremental backups associated with the differential backup will also expire.

When not defined, all differential backups will be retained until the dependent full backups expires.

4.3 Archive retention

pgBackRest allows to define a specific retention policy for archives.

Warning: pgBackRest relies on backup start location to define the archives to clean. In some cases, the oldest archived WAL file in the repository might be older than the first backup start location, in which case it will not be removed until the first backup expires.

4.4 `repo1-retention-archive-type`

The `repo1-retention-archive-type` option can be set to define the backup types counted for WAL retention.

If the `repo1-retention-archive-type` option is set to `full` (default), pgBackRest will keep archived WAL files for the number of full backups as defined by `repo1-retention-archive`.

If the `repo1-retention-archive-type` option is set to `diff` (differential), pgBackRest will keep archives for the number of full and differential backups defined by `repo1-retention-archive`. As an example, if the last backup taken was a full backup, it will be counted as a differential for the purpose of the repository retention.

If the `repo1-retention-archive-type` option is set to `incr` (incremental), pgBackRest will keep archives for the number of full, differential, and incremental backups defined by `repo1-retention-archive`.

Warning: It is recommended to keep the `repo1-retention-archive-type` option with its default, in which case archives will be removed along with full backups.

4.5 `repo1-retention-archive`

The `repo1-retention-archive` option can be set to define the number of backups preserved for continuous WAL retention.

Warning: WAL segments required to make a backup consistent (based on the full and differential backups retention policies) are always retained until the backup expires regardless of how the `repo1-retention-archive` option is configured.

The `repo1-retention-archive` is highly dependent on other options:

- If `repo1-retention-full-type` is set to `time`, then the `repo1-retention-archive` value will default to removing archived WAL files that are older than the oldest full backup retained after satisfying the `repo1-retention-full` setting.
- When `repo1-retention-archive-type` is set to `incr`, the `repo1-retention-archive` must be set.
- If the `repo1-retention-archive` value is not set, and `repo1-retention-full-type` is set to `count` (default), then the `repo1-retention-archive-type` value defines how archives will expire: when `repo1-retention-archive-type` is set to `full` or `diff`, archives expiry will default to the `repo1-retention-full` or `repo1-retention-diff` respectively. This configuration will ensure that the archived WAL files are only removed for corresponding expired backups.

Using the `repo1-retention-archive` setting in conjunction with `repo1-retention-archive-type` can reclaim disk-space by removing extra WAL segments. This configuration is **notrecommended** as it negates the ability to perform point-in-time recovery from the backups with expired WAL.

4.6 Glossary

4.7 pgBackRest

`repo-retention-archive-type` `repo-retention-archive` `repo-retention-diff` `repo-retention-full-type` `repo-retention-full`

RUNNING PGBACKREST LOCALLY ON THE DATABASE HOST'

5.1 Description

- pgBackRest runs locally on database servers and stores repository on remote storage (NFS, S3, Azure or GCS compatible object stores).
- Cron task active on the node where to take backups.
- Manually reconfigure cron task to take backup from another server.

! Use case diagram

5.2 Configuration

5.3 Global Section

The [global] section will contain the:

- repository location;
- recommended settings;
- retention policy.

This section's settings should be the same for all the nodes (primary and standbys).

5.3.1 Mount Points

Set `repo1-type` to **posix** or **cifs** and specify the `repo1-path` location to store backups and archives on a shared space:

```
repo1-type=posix  
repo1-path=/backup_space
```

5.3.2 S3-compatible Object Stores

```
repo1-type=s3
repo1-path=/repo1
repo1-s3-bucket=bucket-name
repo1-s3-endpoint=...
repo1-s3-key=accessKey1
repo1-s3-key-secret=***
repo1-s3-region=...
```

The backups and archives will be stored in `bucket-name/repo1` .

It is possible to automatically retrieve temporary credentials when the AWS instance is associated with an IAM role. Credentials are automatically updated when they are ≤ 5 minutes from expiring.

To do this, instead of setting `repo1-s3-key` and `repo1-s3-key-secret` , use:

```
repo1-s3-key-type=auto
repo1-s3-role=authrole
```

To automatically retrieve web identity credentials, set the `repo1-s3-key-type=web-id` option and define the `AWS_ROLE_ARN` and `AWS_WEB_IDENTITY_TOKEN_FILE` environment variables.

Set the `repo1-s3-kms-key-id` option to enable server-side encryption using the specified AWS key management service key.

5.3.3 Azure-compatible Object Stores

```
repo1-type=azure
repo1-azure-key-type=shared
repo1-path=/repo1
repo1-azure-container=container-name
repo1-azure-account=pgbackrest
repo1-azure-key=***
```

To use the shared access signatures, set the `repo1-azure-key-type` option to `sas` and the `repo1-azure-key` option to the shared access signature token.

5.3.4 GCS-compatible Object Stores

```
repo1-type=gcs
repo1-gcs-key-type=service
repo1-path=/repo1
repo1-gcs-bucket=bucket-name
repo1-gcs-key=/etc/pgbackrest/gcs-key.json
```

`repo1-gcs-key` is a **token** or **service** key file depending on the `repo1-gcs-key-type` option.

5.3.5 Recommended Settings

Use those settings to enable encryption, parallel operations and ensure displaying enough information in the console and in the log file:

```
repol-cipher-type=aes-256-cbc
repol-cipher-pass=FIXME
process-max=FIXME
log-level-console=info
log-level-file=debug
start-fast=y
delta=y
```

Adjust the encryption passphrase and the maximum number of processes to use for compression usage and file transfer. See the ‘Recommended Settings’ page for more details.

5.3.6 Retention Policy

To be able to delete old backups and archives, you have to define a retention policy. Use the following setting to only keep the latest full backup:

```
repol-retention-full=1
```

See the *Retention Policy* page for more options and details.

5.4 Stanza Section

The [demo] stanza section will be different on each database server and should contain:

- the location of the **PostgreSQL** configuration files

```
pg1-path=/var/lib/pgsql/13/data
```

or for **EDBPostgresAdvancedServer**

```
pg1-path=/var/lib/edb/as13/data
pg1-user=enterprisedb
pg1-port=5444
```

- for every other **PostgreSQL** node

```
pg2-host=other_node_hostname_or_ip
pg2-path=/var/lib/pgsql/13/data
```

or for **EDBPostgresAdvancedServer**

```
pg2-host=other_node_hostname_or_ip
pg2-host-user=enterprisedb
pg2-path=/var/lib/edb/as13/data
```

- specific recovery option to set the replication `primary_conninfo` after the restore command is executed:

```
recovery-option=primary_conninfo=host=primary_vip user=replication_user ...
```

5.5 Setup Archiving

Once pgBackRest is configured, set up the database archiver process on each node:

```
# postgresql.conf

archive_mode = on
archive_command = pgbackrest --stanza=demo archive-push %p
```

As changing the `archive_mode` parameter requires a service restart, and changing the `archive_command` only requires a configuration reload, we recommend enabling `archive_mode` with an empty `archive_command` (or pointing to `/bin/true`) when initiating a new database cluster.

See the [Quick Start](#) page to set up a simple **demo** stanza configuration.

Assume the identity of the system user (`postgres` or `enterprisedb`) and initiate the pgBackRest repository from the primary server:

```
$ pgbackrest --stanza=demo stanza-create
```

Check the configuration and the archiving process:

```
$ pgbackrest --stanza=demo check
```

5.6 Backup and Restore

See the **Quickstart** [Backups](#) and [Restore](#) sections for more details about those two commands.

5.7 Glossary

5.8 pgBackRest

Delta log-level-console log-level-file pg-host-user pg-host pg-path pg-port pg-user process-max recovery-option repo-azure-account repo-azure-container repo-azure-key-type repo-azure-key repo-cipher-pass repo-cipher-type repo-path repo-retention-full repo-s3-bucket repo-s3-endpoint repo-s3-key-secret repo-s3-key-type repo-s3-key repo-s3-kms-key-id repo-s3-region repo-s3-role repo-type start-fast

5.9 PostgreSQL

archive_command archive_mode primary_conninfo

RUNNING PGBACKREST FROM A DEDICATED REPOSITORY HOST'

6.1 Description

- pgBackRest runs on a dedicated backup server and stores repository on local or remote storage (NFS, S3, Azure or GCS compatible object stores).
- Cron task active to take backups from the primary database cluster.
- Manually reconfigure cron task to take backup from another server.

! Use case diagram

6.2 Create a Dedicated User on the Repository Host

The `pgbackrest` user is created to own the backups and archives repository. Any user can own the repository, but it is recommended not to use `postgres` or `enterprisedb` to avoid confusion.

```
$ sudo groupadd pgbackrest
$ sudo adduser -g pgbackrest -n pgbackrest
$ sudo chown pgbackrest: /var/log/pgbackrest
```

6.3 Setup Passwordless SSH Connection

pgBackRest requires a passwordless SSH connection to enable communication between the hosts.

The following example SSH commands should succeed without prompting you for a password:

```
# From the PostgreSQL nodes

$ sudo -u postgres ssh pgbackrest@backup-server

# From the EDB Postgres Advanced Server nodes

$ sudo -u enterprisedb ssh pgbackrest@backup-server

# From the repository host

$ sudo -u pgbackrest ssh postgres@postgres-node-1
```

(continues on next page)

(continued from previous page)

```
$ sudo -u pgbackrest ssh postgres@postgres-node-2
$ ...
```

6.4 Configuration

6.5 Backup Server

The repository should be reachable from the backup server. It can be located on the various supported types described in the first use case .

For example, let us create `/backup_space` to store our backups and archives locally:

```
$ sudo mkdir /backup_space
$ sudo chown pgbackrest: /backup_space
```

The `[global]` section of the configuration file will look like this:

```
[global]
repo1-path=/backup_space
repo1-retention-full=2
repo1-cipher-type=aes-256-cbc
repo1-cipher-pass=FIXME
process-max=FIXME
log-level-console=info
log-level-file=debug
start-fast=y
delta=y
```

Adjust the encryption passphrase and the maximum number of processes to use for compression usage and file transfer.

See the ‘Recommended Settings’ pages for more details.

The `[demo]` stanza section will contain specific `pg-` options for each database node.

- the primary database server configuration for **PostgreSQL**

```
pg1-host=postgres-node-1
pg1-path=/var/lib/pgsql/13/data
```

or for **EDBPostgresAdvancedServer**

```
pg1-host=postgres-node-1
pg1-host-user=enterprisedb
pg1-path=/var/lib/edb/as13/data
```

- for every other **PostgreSQL** node

```
pg2-host=other_node_hostname_or_ip
pg2-path=/var/lib/pgsql/13/data
```

or for **EDBPostgresAdvancedServer**

```
pg2-host=other_node_hostname_or_ip
pg2-host-user=enterprisedb
pg2-path=/var/lib/edb/as13/data
```

Define multiple nodes using pg2 , pg3 ,... to reach standby servers only when those servers are available.

6.6 Database server(s)

Update the pgBackRest configuration file on the database server(s):

```
# pgbackrest.conf

[global]
repol-host=backup-server
repol-host-user=pgbackrest
log-level-console=info
log-level-file=debug
delta=y

[demo]
pg1-path=/var/lib/pgsql/13/data
recovery-option=primary_conninfo=host=primary_vip user=replication_user ...
```

The `demo` page for more details.

6.7 Setup Archiving

Once pgBackRest is configured, set up the database archiver process on each node:

```
# postgresql.conf

archive_mode = on
archive_command = pgbackrest --stanza=demo archive-push %p
```

As changing the `archive_mode` parameter requires a service restart, and changing the `archive_command` only requires a configuration reload, we recommend enabling `archive_mode` with an empty `archive_command` (or pointing to `/bin/true`) when initiating a new database cluster.

From the backup server (also called repository host), initiate the pgBackRest repository with the system user we created earlier:

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

Check the configuration and the archiving process:

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

6.8 Backup and Restore

See the **Quickstart** *Backups* and *Restore* sections for more details about those two commands.

The backup command will be executed on the repository host (aka. *backup-server*), while the restore command should be executed on database nodes (aka. *postgres-node-1,...*).

6.9 Glossary

6.10 pgBackRest

Delta log-level-console log-level-file pg-host pg-path pg-port pg-user process-max recovery-option repo-cipher-pass repo-cipher-type repo-host repo-host-user repo-path repo-retention-full start-fast

6.11 PostgreSQL

archive_command archive_mode primary_conninfo

‘MULTIPLE REPOSITORIES’

A key feature of the [v2.33](#) pgBackrest release is support for multiple repositories. This lets you define different behavior for different backup repositories. As an example, you could have a local repository for fast restore that had short retention period (to save space) paired with a remote repository which is larger for cold storage.

This feature introduces a new `--repo` option which can be used to set the repository the command should apply to. All `repo*` options are indexed to enable configuring multiple repositories.

In this section, we will set up a **demo** cluster (using **EDBPostgresAdvancedServer** version 13 on CentOS 7) to demonstrate how this new feature impacts each pgBackRest command.

7.1 Configuration

It must be noted that in this demo, we will set up two local repositories on our database host. For production purposes, it is **recommended** to store each repository on separate storage, and configure at least one remote repository.

```
[global]
# repository details

repo1-path=/var/lib/edb/as13/backups/repo1
repo1-retention-full=1
repo2-path=/var/lib/edb/as13/backups/repo2
repo2-retention-full=1

# general options

process-max=2
log-level-console=info
log-level-file=debug
start-fast=y
delta=y

[demo]
pg1-path=/var/lib/edb/as13/data
pg1-user=enterprisedb
pg1-port=5444
```

For safety reasons, the `--repo` option can't be defined in the configuration file or a warning will be triggered: **WARN: configuration file contains command-line only option 'repo'** . If there is more than one repository configured and the `--repo` option is not specified for a command, the repository with the highest priority order (e.g. `repo1` then `repo2`) will be chosen by default.

The `--repo` option is not required when only `repo1` is configured to maintain backward compatibility. However, when a single repository is configured, it is recommended to use `repo1` in the configuration.

7.2 Initialization

7.3 Stanza Create Command

The `stanza-create` command will automatically operate on all configured repositories:

```
$ pgbackrest --stanza=demo stanza-create
P000 INFO: stanza-create for stanza demo on repo1
P000 INFO: stanza-create for stanza demo on repo2
```

7.4 Check Command

The `check` command will trigger a new WAL segment to be archived and try to push it to all defined repositories:

```
$ pgbackrest --stanza=demo check
P000 INFO: check repo1 configuration (primary)
P000 INFO: check repo2 configuration (primary)
P000 INFO: check repo1 archive for WAL (primary)
P000 INFO: WAL segment ... successfully archived to ... on repo1
P000 INFO: check repo2 archive for WAL (primary)
P000 INFO: WAL segment ... successfully archived to ... on repo2
```

7.5 Archive Push Command

The `archive-push` command will always archive WALs in all configured repositories. Backups will need to be scheduled individually for each repository. In most cases, this is desirable since backup types and retention could vary per repository.

The `archive_command` can still be defined as usual:

```
$ psql -d postgres -c "show archive_command;"
archive_command
-----
pgbackrest --stanza=demo archive-push %p
(1 row)
```

Here is a `DEBUG` extract of the PostgreSQL logs showing the `archive-push` activity:

```
...
P000 DEBUG: storage/storage::storageNewWrite: => {
  type: posix, name: {"/var/lib/edb/as13/backups/repo1/archive/demo/13-1/
  ↳ 000000001000000000/
                                00000000100000000000000005-
  ↳ be9a61a800934879842fa647b0d50385f44e0228.gz"},
  modeFile: 0640, modePath: 0750, createPath: true, syncFile: true, syncPath: true,
  ↳ atomic: true}
(continues on next page)
```

(continued from previous page)

```
...
P000 DEBUG:    storage/storage::storageNewWrite: => {
    type: posix, name: {"var/lib/edb/as13/backups/repo2/archive/demo/13-1/
    ↪000000001000000000/
                        00000000100000000000000005-
    ↪be9a61a800934879842fa647b0d50385f44e0228.gz"},
    modeFile: 0640, modePath: 0750, createPath: true, syncFile: true, syncPath: true,
    ↪atomic: true}
...
P000 INFO: pushed WAL file 00000000100000000000000005 to the archive
```

The archive-push command will try to push the WAL archive to all reachable repositories. The idea is to archive to as many repositories as possible even if we still need to throw an error to PostgreSQL to prevent it from removing the WAL file.

```
P000 INFO: archive-push command begin: [pg_wal/00000000100000000000000006] ...
...
P000 INFO: archive-push command begin: [pg_wal/00000000100000000000000007] ...
ERROR: [104]: archive-push command encountered error(s):
    repo2: [PathOpenError] unable to list file info for path ...: [13] Permission
    ↪denied
P000 INFO: archive-push command end: aborted with exception [104]
DETAIL: The failed archive command was: pgbackrest --stanza=demo archive-push pg_wal/
    ↪00000000100000000000000007
WARNING: archiving write-ahead log file "00000000100000000000000007" failed too many
    ↪times, will try again later
```

The PostgreSQL archiver process should then report an error:

```
$ ps -o pid,cmd fx |grep archiver
4440 \_ postgres: archiver failed on 00000000100000000000000007
```

The next WAL segments should not then be archived:

```
$ psql -d postgres -c "SELECT pg_create_restore_point(generate some activity); SELECT pg_
    ↪switch_wal();"
pg_switch_wal
-----
0/80001C8
(1 row)

$ ls as13/data/pg_wal/archive_status/ |grep ".ready"
00000000100000000000000007.ready
00000000100000000000000008.ready
```

This becomes very handy by adding archive-async=y to the configuration in order to use asynchronous archiving processes within pgBackRest itself. Even if PostgreSQL archiver process is still stuck, the archives will reach the working repositories:

```
$ $ ls /var/lib/edb/as13/backups/repo1/archive/demo/13-1/000000001000000000/
00000000100000000000000005-be9a61a800934879842fa647b0d50385f44e0228.gz
00000000100000000000000006-7108296955e1208c93447dcfc5712ce59af97907.gz
00000000100000000000000007-f1b6fcdd23e96a19a7a89dc92613674c2316aa49.gz
```

(continues on next page)

(continued from previous page)

```

00000000100000000000000008-791c44d39ef772d58e51d64c8e3589231d3916a4.gz
00000000100000000000000009-47241214eb3fd5e693eed778caa7b304a778ecc2.gz

$ ls /var/lib/edb/as13/backups/repo2/archive/demo/13-1/000000001000000000/
00000000100000000000000005-be9a61a800934879842fa647b0d50385f44e0228.gz
00000000100000000000000006-7108296955e1208c93447dcfc5712ce59af97907.gz

```

Let us unblock the archiver process (and remove asynchronous archiving) before going further:

```

$ $ ps -o pid,cmd fx |grep archiver
4440 \ postgres: archiver last was 000000001000000000000000

```

7.6 Backups

7.7 Backup Command

Let us take a few backups:

```

$ pgbackrest backup --stanza=demo --type=full
P00 INFO: backup command begin ...
P00 INFO: repo option not specified, defaulting to repo1
P00 INFO: execute non-exclusive pg_start_backup(): backup begins after the requested_
↳immediate checkpoint completes
P00 INFO: backup start archive = 00000000100000000000000000C, lsn = 0/C000060
P00 INFO: full backup size = 50.4MB
P00 INFO: execute non-exclusive pg_stop_backup() and wait for all WAL segments to_
↳archive
P00 INFO: backup stop archive = 00000000100000000000000000C, lsn = 0/C000138
P00 INFO: check archive for segment(s)_
↳00000000100000000000000000C:00000000100000000000000000C
P00 INFO: new backup label = 20210419-142212F
P00 INFO: backup command end: completed successfully

$ pgbackrest backup --stanza=demo --type=full --repo=2
P00 INFO: backup command begin ...
P00 INFO: execute non-exclusive pg_start_backup(): backup begins after the requested_
↳immediate checkpoint completes
P00 INFO: backup start archive = 00000000100000000000000000E, lsn = 0/E000028
P00 INFO: full backup size = 50.5MB
P00 INFO: execute non-exclusive pg_stop_backup() and wait for all WAL segments to_
↳archive
P00 INFO: backup stop archive = 00000000100000000000000000E, lsn = 0/E000138
P00 INFO: check archive for segment(s)_
↳00000000100000000000000000E:00000000100000000000000000E
P00 INFO: new backup label = 20210419-142414F
P00 INFO: backup command end: completed successfully

$ pgbackrest backup --stanza=demo --type=incr --repo=1
P00 INFO: backup command begin ...

```

(continues on next page)

(continued from previous page)

```

P00 INFO: last backup label = 20210419-142212F, version = ...
P00 INFO: execute non-exclusive pg_start_backup(): backup begins after the requested_
↳immediate checkpoint completes
P00 INFO: backup start archive = 0000000100000000000000010, lsn = 0/10000028
P00 INFO: incr backup size = 50.5MB
P00 INFO: execute non-exclusive pg_stop_backup() and wait for all WAL segments to_
↳archive
P00 INFO: backup stop archive = 0000000100000000000000010, lsn = 0/10000100
P00 INFO: check archive for segment(s)_
↳0000000100000000000000010:0000000100000000000000010
P00 INFO: new backup label = 20210419-142212F_20210419-142502I
P00 INFO: backup command end: completed successfully

$ pgbackrest backup --stanza=demo --type=incr --repo=2
P00 INFO: backup command begin ...
P00 INFO: last backup label = 20210419-142414F, version = ...
P00 INFO: execute non-exclusive pg_start_backup(): backup begins after the requested_
↳immediate checkpoint completes
P00 INFO: backup start archive = 0000000100000000000000012, lsn = 0/12000028
P00 INFO: incr backup size = 50.6MB
P00 INFO: execute non-exclusive pg_stop_backup() and wait for all WAL segments to_
↳archive
P00 INFO: backup stop archive = 0000000100000000000000012, lsn = 0/12000100
P00 INFO: check archive for segment(s)_
↳0000000100000000000000012:0000000100000000000000012
P00 INFO: new backup label = 20210419-142414F_20210419-142556I
P00 INFO: backup command end: completed successfully

```

Here, we alternated full and incr backups in each repository.

7.8 Info Command

```

$ pgbackrest info --stanza=demo
stanza: demo
  status: ok
  cipher: none

db (current)
  wal archive min/max (13): 000000010000000000000002/000000010000000000000012

  full backup: 20210419-142212F
    timestamp start/stop: 2021-04-19 14:22:12 / 2021-04-19 14:22:26
    wal start/stop: 000000010000000000000000C / 000000010000000000000000C
    database size: 50.4MB, database backup size: 50.4MB
    repo1: backup set size: 8MB, backup size: 8MB

  full backup: 20210419-142414F
    timestamp start/stop: 2021-04-19 14:24:14 / 2021-04-19 14:24:26
    wal start/stop: 000000010000000000000000E / 000000010000000000000000E
    database size: 50.5MB, database backup size: 50.5MB

```

(continues on next page)

(continued from previous page)

```

repo2: backup set size: 8MB, backup size: 8MB

incr backup: 20210419-142212F_20210419-142502I
timestamp start/stop: 2021-04-19 14:25:02 / 2021-04-19 14:25:04
wal start/stop: 00000000100000000000000010 / 00000000100000000000000010
database size: 50.5MB, database backup size: 613.4KB
repo1: backup set size: 8MB, backup size: 26.3KB
backup reference list: 20210419-142212F

incr backup: 20210419-142414F_20210419-142556I
timestamp start/stop: 2021-04-19 14:25:56 / 2021-04-19 14:25:57
wal start/stop: 00000000100000000000000012 / 00000000100000000000000012
database size: 50.6MB, database backup size: 661.5KB
repo2: backup set size: 8MB, backup size: 28KB
backup reference list: 20210419-142414F

```

The default order will sort backups by dates mixing the repositories. It might be confusing to find the backups depending on each other.

We can then split this view per repository:

```

$ pgbackrest info --stanza=demo --repo=1
stanza: demo
  status: ok
  cipher: none

db (current)
  wal archive min/max (13): 000000001000000000000000C/00000000100000000000000012

  full backup: 20210419-142212F
    timestamp start/stop: 2021-04-19 14:22:12 / 2021-04-19 14:22:26
    wal start/stop: 000000001000000000000000C / 00000000100000000000000000C
    database size: 50.4MB, database backup size: 50.4MB
    repo1: backup set size: 8MB, backup size: 8MB

  incr backup: 20210419-142212F_20210419-142502I
    timestamp start/stop: 2021-04-19 14:25:02 / 2021-04-19 14:25:04
    wal start/stop: 00000000100000000000000010 / 00000000100000000000000010
    database size: 50.5MB, database backup size: 613.4KB
    repo1: backup set size: 8MB, backup size: 26.3KB
    backup reference list: 20210419-142212F

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

db (current)
  wal archive min/max (13): 000000001000000000000000E/00000000100000000000000012

  full backup: 20210419-142414F
    timestamp start/stop: 2021-04-19 14:24:14 / 2021-04-19 14:24:26
    wal start/stop: 000000001000000000000000E / 000000001000000000000000E

```

(continues on next page)

(continued from previous page)

```

database size: 50.5MB, database backup size: 50.5MB
repo2: backup set size: 8MB, backup size: 8MB

incr backup: 20210419-142414F_20210419-142556I
timestamp start/stop: 2021-04-19 14:25:56 / 2021-04-19 14:25:57
wal start/stop: 000000001000000000000000000012 / 000000001000000000000000000012
database size: 50.6MB, database backup size: 661.5KB
repo2: backup set size: 8MB, backup size: 28KB
backup reference list: 20210419-142414F

```

The ‘**walarchivemin/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. There may be gaps due to archive retention policies or other reasons.

Let us break the first repository by removing its content:

```

$ pgbackrest --stanza=demo stanza-delete --repo=1 --force
P00 INFO: stanza-delete command end: completed successfully

$ pgbackrest --stanza=demo stanza-create
P00 INFO: stanza-create for stanza demo on repo1
P00 INFO: stanza-create for stanza demo on repo2
P00 INFO: stanza demo already exists on repo2 and is valid
P00 INFO: stanza-create command end: completed successfully

```

If multiple repositories are configured, then a status of **mixed** indicates that the stanza is not in a healthy state for one or more of the repositories. In this case, the state of the stanza will be detailed in additional lines per repository.

```

$ pgbackrest info --stanza=demo
stanza: demo
  status: mixed
    repo1: error (no valid backups)
    repo2: ok
  cipher: none

db (current)
  wal archive min/max (13): 00000000100000000000000000000E/000000001000000000000000000012

  full backup: 20210419-142414F
    timestamp start/stop: 2021-04-19 14:24:14 / 2021-04-19 14:24:26
    wal start/stop: 00000000100000000000000000000E / 00000000100000000000000000000E
    database size: 50.5MB, database backup size: 50.5MB
    repo2: backup set size: 8MB, backup size: 8MB

  incr backup: 20210419-142414F_20210419-142556I
    timestamp start/stop: 2021-04-19 14:25:56 / 2021-04-19 14:25:57
    wal start/stop: 000000001000000000000000000012 / 000000001000000000000000000012
    database size: 50.6MB, database backup size: 661.5KB
    repo2: backup set size: 8MB, backup size: 28KB
    backup reference list: 20210419-142414F

```

This state can be resolved by taking a new backup:

```

$ pgbackrest backup --stanza=demo --type=full --repo=1
P00 INFO: backup command begin ...
P00 INFO: execute non-exclusive pg_start_backup(): backup begins after the requested_
↳immediate checkpoint completes
P00 INFO: backup start archive = 00000001000000000000000014, lsn = 0/14000028
P00 INFO: full backup size = 50.6MB
P00 INFO: execute non-exclusive pg_stop_backup() and wait for all WAL segments to_
↳archive
P00 INFO: backup stop archive = 00000001000000000000000014, lsn = 0/14000138
P00 INFO: check archive for segment(s)_
↳00000001000000000000000014:00000001000000000000000014
P00 INFO: new backup label = 20210419-143400F
P00 INFO: backup command end: completed successfully

```

7.9 Restore

Let us initiate a situation with some data, backups and restore point:

```

$ createdb test
$ psql -d test -c "CREATE TABLE t1(id int);"
CREATE TABLE
$ psql -d test -c "INSERT INTO t1 VALUES (1);"
INSERT 0 1

$ pgbackrest backup --stanza=demo --type=full --repo=1
P00 INFO: backup command begin ...
P00 INFO: execute non-exclusive pg_start_backup(): backup begins after the requested_
↳immediate checkpoint completes
P00 INFO: backup start archive = 00000001000000000000000016, lsn = 0/16000028
P00 INFO: full backup size = 63.0MB
P00 INFO: execute non-exclusive pg_stop_backup() and wait for all WAL segments to_
↳archive
P00 INFO: backup stop archive = 00000001000000000000000016, lsn = 0/16000138
P00 INFO: check archive for segment(s)_
↳00000001000000000000000016:00000001000000000000000016
P00 INFO: new backup label = 20210419-143643F
P00 INFO: backup command end: completed successfully
P00 INFO: expire command begin ...
P00 INFO: repo1: expire full backup 20210419-143400F
P00 INFO: repo1: remove expired backup 20210419-143400F
P00 INFO: expire command end: completed successfully

$ psql -d test -c "INSERT INTO t1 VALUES (2);"
INSERT 0 1
$ pgbackrest backup --stanza=demo --type=full --repo=2
P00 INFO: backup command begin ...
P00 INFO: execute non-exclusive pg_start_backup(): backup begins after the requested_
↳immediate checkpoint completes
P00 INFO: backup start archive = 00000001000000000000000018, lsn = 0/18000028
P00 INFO: full backup size = 63MB
P00 INFO: execute non-exclusive pg_stop_backup() and wait for all WAL segments to_
↳archive

```

(continues on next page)

(continued from previous page)

```

P00 INFO: backup stop archive = 000000010000000000000018, lsn = 0/18000138
P00 INFO: check archive for segment(s)
    ↳ 000000010000000000000018:000000010000000000000018
P00 INFO: new backup label = 20210419-143749F
P00 INFO: backup command end: completed successfully
P00 INFO: expire command begin ...
P00 INFO: repo2: expire full backup set 20210419-142414F, 20210419-142414F_20210419-
    ↳ 142556I
P00 INFO: repo2: remove expired backup 20210419-142414F_20210419-142556I
P00 INFO: repo2: remove expired backup 20210419-142414F
P00 INFO: expire command end: completed successfully

$ psql -d postgres -c "select pg_create_restore_point(RP1);"
pg_create_restore_point
-----
0/190000C8
(1 row)

$ psql -d postgres -Atc "select current_timestamp,current_setting(datestyle),txid_
    ↳ current();"
19-APR-21 14:38:55.313716 +00:00|Redwood, SHOW_TIME|1183

$ psql -d test -c "INSERT INTO t1 VALUES (3); SELECT pg_switch_wal();"
pg_switch_wal
-----
0/19000238
(1 row)

$ pgbackrest info --stanza=demo
stanza: demo
    status: ok
    cipher: none

    db (current)
        wal archive min/max (13): 000000010000000000000016/000000010000000000000019

        full backup: 20210419-143643F
            timestamp start/stop: 2021-04-19 14:36:43 / 2021-04-19 14:37:01
            wal start/stop: 000000010000000000000016 / 000000010000000000000016
            database size: 63.0MB, database backup size: 63.0MB
            repo1: backup set size: 10MB, backup size: 10MB

        full backup: 20210419-143749F
            timestamp start/stop: 2021-04-19 14:37:49 / 2021-04-19 14:38:03
            wal start/stop: 000000010000000000000018 / 000000010000000000000018
            database size: 63MB, database backup size: 63MB
            repo2: backup set size: 10MB, backup size: 10MB

```

As you can see in the example above, the backup command automatically triggered the expire command on the repository when we created a backup. Since this command is operating on one repository only, it would not expire anything in the other repositories until the backup or expire command is run for that repository.

Since the default retention policy is based on a number of backups, it is expected that only new backups are affecting

the backups and archives to expire.

7.10 Restore Command

The restore command automatically defaults to selecting the latest backup from the first repository where backups exist. The order in which the repositories are checked is dictated by order of the repositories as configured in the `pgbackrest.conf` (e.g. `repo1` will be checked before `repo2`). To restore from from a specific repository, the `--repo` option can be used.

PITR can be performed by specifying `--type=time` and specifying the target time with `--target`. If a backup is not specified via the `--set` option, then the configured repositories will be checked for a backup that contains the requested time. If no backup can be found, the latest backup from the first repository containing backups will be used.

```
$ mkdir /tmp/restored_data
$ pgbackrest restore --stanza=demo --target="RP1" --type=name --no-delta --pg1-path=/tmp/
↪restored_data
P00 INFO: repo1: restore backup set 20210419-143643F

$ rm -rf /tmp/restored_data/*
$ pgbackrest restore --stanza=demo --target="2021-04-19 14:38:55.313716+00:00" --
↪type=time --no-delta --pg1-path=/tmp/restored_data
HINT: time format must be YYYY-MM-DD HH:MM:SS with optional msec and optional timezone.
↪(+/- HH or HHMM or HH:MM)
- if timezone is omitted, local time is assumed (for UTC use +00)
P00 INFO: repo1: restore backup set 20210419-143643F

$ rm -rf /tmp/restored_data/*
$ pgbackrest restore --stanza=demo --target="RP1" --type=name --no-delta --pg1-path=/tmp/
↪restored_data --repo=2
P00 INFO: repo2: restore backup set 20210419-143749F
```

Even if the backup in **repo2** is newer, the first found match is preserved. Let's take a new backup in **repo1** to check if pgBackRest will auto-select the backup in **repo2**:

```
$ pgbackrest backup --stanza=demo --type=full --repo=1
P00 INFO: backup command begin ...
P00 INFO: execute non-exclusive pg_start_backup(): backup begins after the requested.
↪immediate checkpoint completes
P00 INFO: backup start archive = 000000010000000000000001B, lsn = 0/1B000028
P00 INFO: full backup size = 63MB
P00 INFO: execute non-exclusive pg_stop_backup() and wait for all WAL segments to.
↪archive
P00 INFO: backup stop archive = 000000010000000000000001B, lsn = 0/1B000138
P00 INFO: check archive for segment(s).
↪000000010000000000000001B:000000010000000000000001B
P00 INFO: new backup label = 20210419-144656F
P00 INFO: backup command end: completed successfully
P00 INFO: expire command begin ...
P00 INFO: repo1: expire full backup 20210419-143643F
P00 INFO: repo1: remove expired backup 20210419-143643F
P00 INFO: expire command end: completed successfully
```

(continues on next page)

(continued from previous page)

```

$ pgbackrest info --stanza=demo
stanza: demo
    status: ok
    cipher: none

db (current)
    wal archive min/max (13): 0000000100000000000000018/000000010000000000000001B

    full backup: 20210419-143749F
        timestamp start/stop: 2021-04-19 14:37:49 / 2021-04-19 14:38:03
        wal start/stop: 0000000100000000000000018 / 0000000100000000000000018
        database size: 63MB, database backup size: 63MB
        repo2: backup set size: 10MB, backup size: 10MB

    full backup: 20210419-144656F
        timestamp start/stop: 2021-04-19 14:46:56 / 2021-04-19 14:47:10
        wal start/stop: 000000010000000000000001B / 000000010000000000000001B
        database size: 63MB, database backup size: 63MB
        repo1: backup set size: 10MB, backup size: 10MB

$ rm -rf /tmp/restored_data/*
$ pgbackrest restore --stanza=demo --target="2021-04-19 14:38:55.313716+00:00" --
↪type=time --no-delta --pg1-path=/tmp/restored_data
P00 INFO: repo2: restore backup set 20210419-143749F

```

The recovery process may be complex and tricky depending on the target, and therefore, the `info` command can really be helpful to determine the repository to restore from.

7.11 Archive Get Command

When multiple repositories are configured, WAL will be fetched from the repositories in priority order (e.g. `repo1`, `repo2`, etc.). In general it is better if faster storage has higher priority. The command can operate on a single repository by specifying it with the `--repo` option.

```

root# systemctl stop edb-as-13.service
$ mv /var/lib/edb/as13/data /var/lib/edb/as13/data.orig
$ mkdir -m 700 /var/lib/edb/as13/data

$ pgbackrest restore --stanza=demo --target="2021-04-19 14:38:55.313716+00:00" --
↪type=time --no-delta --target-action=promote
P00 INFO: restore command begin ...
P00 INFO: write updated /var/lib/edb/as13/data/postgresql.auto.conf
P00 INFO: restore global/pg_control (performed last to ensure aborted restores cannot
↪be started)
P00 INFO: restore command end: completed successfully

$ cat /var/lib/edb/as13/data/postgresql.auto.conf|grep restore_command
restore_command = pgbackrest --stanza=demo archive-get %f "%p"

root# systemctl start edb-as-13.service

```

(continues on next page)

(continued from previous page)

```
$ cat /var/lib/edb/as13/data/log/edb-2021-04-19_145147.log
...
LOG: starting point-in-time recovery to 2021-04-19 14:38:55.313716+00
P00 INFO: found 00000001000000000000000018 in the repo2: 13-1 archive
P00 INFO: found 00000001000000000000000019 in the repo2: 13-1 archive
LOG: recovery stopping before commit of transaction 1183, time 2021-04-19 14:38:55.
    ↪ 314118+00
LOG: redo done at 0/19000100
P00 INFO: unable to find 00000002.history in the archive
LOG: selected new timeline ID: 2
LOG: archive recovery complete
P00 INFO: pushed WAL file 00000002.history to the archive

$ ls /var/lib/edb/as13/backups/repo*/archive/demo/13-1/00000002.history
/var/lib/edb/as13/backups/repo1/archive/demo/13-1/00000002.history
/var/lib/edb/as13/backups/repo2/archive/demo/13-1/00000002.history
```

PostgreSQL found the WALs needed for recovery in our **repo2**, picked a new timeline and pushed the history file to the repositories.

Whenever a new timeline is created, PostgreSQL creates a timeline history file that shows which timeline it branched off from and when. These history files are necessary to allow the system to pick the right WAL segment files when recovering from a backup that contains multiple timelines.

```
root# systemctl stop edb-as-13.service
$ rm -rf /var/lib/edb/as13/data/*
$ rm -rf /var/lib/edb/as13/backups/repo1/archive/demo/13-1/00000002.history
$ pgbackrest restore --stanza=demo --target="2021-04-19 14:38:55.313716+00:00" --
    ↪ type=time --target-timeline=current --no-delta --target-action=promote
P00 INFO: restore command begin ...
P00 INFO: repo2: restore backup set 20210419-143749F
P00 INFO: write updated /var/lib/edb/as13/data/postgresql.auto.conf
P00 INFO: restore global/pg_control (performed last to ensure aborted restores cannot
    ↪ be started)
P00 INFO: restore command end: completed successfully

root# systemctl start edb-as-13.service
$ cat /var/lib/edb/as13/data/log/edb-2021-04-19_145721.log
P00 INFO: found 00000002.history in the repo2: 13-1 archive
P00 INFO: unable to find 00000003.history in the archive
LOG: selected new timeline ID: 3
P00 INFO: pushed WAL file 00000003.history to the archive
```

As shown above, since the history file is stored within the same repository as the restored backup set, PostgreSQL is able to pick a new and accurate timeline.

Let us now retry after moving the history file to **repo1**.

```
root# systemctl stop edb-as-13.service
$ rm -rf /var/lib/edb/as13/data/*
$ mv /var/lib/edb/as13/backups/repo2/archive/demo/13-1/00000002.history /var/lib/edb/
    ↪ as13/backups/repo1/archive/demo/13-1/00000002.history
$ pgbackrest restore --stanza=demo --target="2021-04-19 14:38:55.313716+00:00" --
    ↪ type=time --target-timeline=current --no-delta --target-action=promote
```

(continues on next page)

(continued from previous page)

```
root# systemctl start edb-as-13.service
$ cat /var/lib/edb/as13/data/log/edb-2021-04-19_150037.log
P00 INFO: found 00000002.history in the repo1: 13-1 archive
P00 INFO: found 00000003.history in the repo1: 13-1 archive
P00 INFO: unable to find 00000004.history in the archive
LOG: selected new timeline ID: 4
P00 INFO: pushed WAL file 00000004.history to the archive
```

As we can see, both 00000002.history and 00000003.history have been found in **repo1** so PostgreSQL could pick the next timeline correctly, even if the restored backup set came from **repo2**.

When using multiple repositories, the `archive-get` command will tolerate gaps in one repository (due to lack of disk space e.g.) mainly because it will still be able to find the missing files in the other repository.

‘TLS CONNECTIONS’

8.1 Description

The TLS server is an alternative to using SSH for protocol connections to remote hosts. It is supported since the [2.37](#) release.

In this section, we will examine a configuration used to run pgBackRest from a dedicated repository host like described in the Use Case 2 , but using TLS rather than SSH connections.

We will set up a **demo** cluster (using **EDBPostgresAdvancedServer** version 14 on Rocky Linux 8) where the repository host will be called **backup-srv** and the 3 Postgres nodes in *Streaming Replication*: **pg1-srv** , **pg2-srv** , **pg3-srv** .

8.2 Create a Dedicated User on the Repository Host

The **pgbackrest** user is created to own the backups and archives repository. Any user can own the repository, but it is recommended not to use **postgres** or **enterprisedb** to avoid confusion.

Create the user and the repository location on **backup-srv** :

```
$ sudo groupadd pgbackrest
$ sudo adduser -g pgbackrest -n pgbackrest
$ sudo chown pgbackrest: /var/log/pgbackrest
```

The repository should be reachable from the backup server. It can be located on the various supported types described in the first use case .

For example, let us create **/backup_space** to store our backups and archives locally:

```
$ sudo mkdir /backup_space
$ sudo chown pgbackrest: /backup_space
```

8.3 Certificates generation

The TLS server implementation relies on certificates.

In this example, we will: * create our own root certificate authority (CA), * generate keys and certificates for the Postgres nodes and for the backup server.

```
$ mkdir certs && cd certs
$ openssl req -new -x509 -days 365 -nodes -out ca.crt -keyout ca.key -subj "/CN=root-ca"
$ openssl req -new -nodes -out backup-srv.csr -keyout backup-srv.key -subj "/CN=backup-
↪srv"
$ openssl req -new -nodes -out pg1-srv.csr -keyout pg1-srv.key -subj "/CN=pg1-srv"
$ openssl req -new -nodes -out pg2-srv.csr -keyout pg2-srv.key -subj "/CN=pg2-srv"
$ openssl req -new -nodes -out pg3-srv.csr -keyout pg3-srv.key -subj "/CN=pg3-srv"
$ openssl x509 -req -in backup-srv.csr -days 365 -CA ca.crt -CAkey ca.key -
↪CAcreateserial -out backup-srv.crt
$ openssl x509 -req -in pg1-srv.csr -days 365 -CA ca.crt -CAkey ca.key -CAcreateserial -
↪out pg1-srv.crt
$ openssl x509 -req -in pg2-srv.csr -days 365 -CA ca.crt -CAkey ca.key -CAcreateserial -
↪out pg2-srv.crt
$ openssl x509 -req -in pg3-srv.csr -days 365 -CA ca.crt -CAkey ca.key -CAcreateserial -
↪out pg3-srv.crt
$ rm *.csr
```

Then, you have to deploy it on each server (ca.crt + server_name.crt + server_name.key).

Here is an example using a shared folder between the hosts:

```
# on backup-srv

$ sudo -u pgbackrest mkdir ~pgbackrest/certs
$ sudo cp `hostname`. * ~pgbackrest/certs
$ sudo cp ca.crt ~pgbackrest/certs
$ sudo chown -R pgbackrest: ~pgbackrest/certs
$ sudo chmod -R og-rwx ~pgbackrest/certs

# on pg1-srv, pg2-srv and pg3-srv

$ sudo -u enterprisedb mkdir ~enterprisedb/certs
$ sudo cp `hostname`. * ~enterprisedb/certs
$ sudo cp ca.crt ~enterprisedb/certs
$ sudo chown -R enterprisedb: ~enterprisedb/certs
$ sudo chmod -R og-rwx ~enterprisedb/certs
```

8.4 Configuration

We will now prepare the configuration for our stanza called `demo`.

Update the **backup-srv** pgBackRest configuration file:

```
# /etc/pgbackrest.conf

[global]
# repo details

repol-path=/backup_space
repol-retention-full=2

# general options

process-max=2
log-level-console=info
log-level-file=debug
start-fast=y
delta=y

# tls server options

tls-server-address=*
tls-server-cert-file=/home/pgbackrest/certs/backup-srv.crt
tls-server-key-file=/home/pgbackrest/certs/backup-srv.key
tls-server-ca-file=/home/pgbackrest/certs/ca.crt
tls-server-auth=pg1-srv=demo
tls-server-auth=pg2-srv=demo
tls-server-auth=pg3-srv=demo

[demo]
pg1-host=pg1-srv
pg1-host-user=enterprisedb
pg1-path=/var/lib/edb/as14/data
pg1-host-type=tls
pg1-host-cert-file=certs/backup-srv.crt
pg1-host-key-file=certs/backup-srv.key
pg1-host-ca-file=certs/ca.crt
```

Update the **pg1-srv**, **pg2-srv** and **pg3-srv** pgBackRest configuration file:

```
# /etc/pgbackrest.conf

[global]
repol-host=backup-srv
repol-host-user=pgbackrest
repol-host-type=tls
repol-host-cert-file=/var/lib/edb/certs/pg1-srv.crt
repol-host-key-file=/var/lib/edb/certs/pg1-srv.key
repol-host-ca-file=/var/lib/edb/certs/ca.crt
```

(continues on next page)

(continued from previous page)

```
# general options

process-max=2
log-level-console=info
log-level-file=debug

# tls server options

tls-server-address=*
tls-server-cert-file=/var/lib/edb/certs/pg1-srv.crt
tls-server-key-file=/var/lib/edb/certs/pg1-srv.key
tls-server-ca-file=/var/lib/edb/certs/ca.crt
tls-server-auth=backup-srv=demo

[demo]
pg1-path=/var/lib/edb/as14/data
pg1-user=enterprisedb
pg1-port=5444
```

(Remark: obviously, adjust *pg1-srv* by *pg2-srv* on **pg2-srv** and *pg3-srv* on **pg3-srv** .)

Clients are authorized on the server by verifying their certificate and checking the certificate CN (*Common Name*) against a list on the server configured with the `tls-server-auth` . A CN can be authorized for as many stanzas as needed by repeating the `tls-server-auth` option, or for all stanzas by specifying `tls-server-auth=*` .

The idea is to have a TLS server running on each host to serve the request coming from the other. In example, the `backup` command running on the repository host will act as client for the TLS server running on the Postgres node. The `archive-push` command running on the Postgres node will act as client for the TLS server running on the repository host.

The *EDB Postgres Advanced Server* command can be used to start the TLS server and will run until terminated by the SIGINT signal (Control+C).

If not done by the PGDG package already, create a service file on the **backup-srv** server:

```
# /etc/systemd/system/pgbackrest.service

[Unit]
Description=pgBackRest Server
After=network.target
StartLimitIntervalSec=0

[Service]
Type=simple
User=pgbackrest
Restart=always
RestartSec=1
ExecStart=/usr/bin/pgbackrest server
ExecStartPost=/bin/sleep 3
ExecStartPost=/bin/bash -c "[ ! -z $MAINPID ]"
ExecReload=/bin/kill -HUP $MAINPID

[Install]
WantedBy=multi-user.target
```

Start the server and check it's running:

```
$ sudo systemctl daemon-reload
$ sudo systemctl enable pgbackrest
$ sudo systemctl start pgbackrest
$ pgbackrest server-ping
```

The `server-ping` command serves as an aliveness check only since no authentication is attempted.

Don't forget to terminate/restart the process when the `tls-*` configuration is changed.

Now, apply the same steps (service configuration and start) on the `pg1-srv`, `pg2-srv` and `pg3-srv` using this configuration:

```
# /etc/systemd/system/pgbackrest.service

[Unit]
Description=pgBackRest Server
After=network.target
StartLimitIntervalSec=0

[Service]
Type=simple
User=enterprisedb
Restart=always
RestartSec=1
ExecStart=/usr/bin/pgbackrest server
ExecStartPost=/bin/sleep 3
ExecStartPost=/bin/bash -c "[ ! -z $MAINPID ]"
ExecReload=/bin/kill -HUP $MAINPID

[Install]
WantedBy=multi-user.target
```

Now, let us configure Postgres archiving in the `postgresql.conf` file (on `pg1-srv`):

```
listen_addresses = *
archive_mode = on
archive_command = pgbackrest --stanza=demo archive-push %p
```

The Postgres instance must be restarted after making these changes and before performing a backup.

Let us finally create the stanza and check the configuration on `backup-srv`:

```
$ sudo -iu pgbackrest pgbackrest --stanza=demo stanza-create
...
P000 INFO: stanza-create command end: completed successfully

$ sudo -iu pgbackrest pgbackrest --stanza=demo check
...
P000 INFO: check command end: completed successfully
```

8.5 Perform a backup

If everything is working correctly, we can now take our first backup:

```
$ sudo -iu pgbackrest pgbackrest --stanza=demo --type=full backup
P00 INFO: backup command begin 2.38: ..
P00 INFO: execute non-exclusive pg_start_backup(): backup begins after the requested
      immediate checkpoint.
↪ completes
P00 INFO: backup start archive = 000000010000000000000004, lsn = 0/4000060
P00 INFO: check archive for prior segment 000000010000000000000003
P00 INFO: execute non-exclusive pg_stop_backup() and wait for all WAL segments to
↪ archive
P00 INFO: backup stop archive = 000000010000000000000004, lsn = 0/4000138
P00 INFO: check archive for segment(s)
↪ 000000010000000000000004:000000010000000000000004
P00 INFO: new backup label = 20220308-143110F
P00 INFO: full backup size = 56.5MB, file total = 1691
P00 INFO: backup command end: completed successfully
```

The info command can then be executed from any server where pgBackRest is correctly configured:

```
# From the Postgres node

$ sudo -iu enterprisedb pgbackrest --stanza=demo info
stanza: demo
  status: ok
  cipher: none

  db (current)
    wal archive min/max (14): 000000010000000000000002/000000010000000000000004

    full backup: 20220308-143110F
      timestamp start/stop: 2022-03-08 14:31:10 / 2022-03-08 14:31:20
      wal start/stop: 000000010000000000000004 / 000000010000000000000004
      database size: 56.5MB, database backup size: 56.5MB
      repo1: backup set size: 9.1MB, backup size: 9.1MB

# From the backup server

$ sudo -iu pgbackrest pgbackrest --stanza=demo info
stanza: demo
  status: ok
  cipher: none

  db (current)
    wal archive min/max (14): 000000010000000000000002/000000010000000000000004

    full backup: 20220308-143110F
      timestamp start/stop: 2022-03-08 14:31:10 / 2022-03-08 14:31:20
      wal start/stop: 000000010000000000000004 / 000000010000000000000004
      database size: 56.5MB, database backup size: 56.5MB
      repo1: backup set size: 9.1MB, backup size: 9.1MB
```

8.6 Prepare the servers for Streaming Replication

On **pg1-srv**, create a specific user for the replication:

```
$ sudo -iu enterprisedb psql -d postgres
postgres=# CREATE ROLE replic_user WITH LOGIN REPLICATION PASSWORD mypwd;
```

Configure `pg_hba.conf`:

```
host    replication    replic_user    pg2-srv    scram-sha-256
host    replication    replic_user    pg3-srv    scram-sha-256
```

Reload configuration:

```
$ sudo systemctl reload edb-as-14.service
```

Configure `~enterprisedb/.pgpass` on **pg2-srv** and **pg3-srv**:

```
$ sudo -iu enterprisedb
$ echo "*:*:replication:replic_user:mypwd" >> ~enterprisedb/.pgpass
$ chown enterprisedb: ~enterprisedb/.pgpass
$ chmod 0600 ~enterprisedb/.pgpass
```

8.7 Setup the standby servers

Adjust `/etc/pgbackrest.conf` on **pg2-srv** and **pg3-srv** to add the `recovery-option`. The idea is to automatically configure the *Streaming Replication* connection string with the restore command.

```
recovery-option=primary_conninfo=host=pg1-srv user=replic_user
```

Then, make sure the configuration is correct by executing the `info` command. It should print the same output as above.

Restore the backup taken from the **pg1-srv** server on **pg2-srv** and **pg3-srv**:

```
$ sudo -iu enterprisedb pgbackrest --stanza=demo --type=standby restore
P00 INFO: restore command begin 2.38: ...
P00 INFO: repo1: restore backup set 20220308-143110F, recovery will start at ...
P00 INFO: write updated /var/lib/edb/as14/data/postgresql.auto.conf
P00 INFO: restore global/pg_control
(performed last to ensure aborted restores cannot be started)
P00 INFO: restore size = 56.5MB, file total = 1691
P00 INFO: restore command end: completed successfully
```

The restore will add extra information to the `postgresql.auto.conf` file:

```
# Recovery settings generated by pgBackRest restore on ...

primary_conninfo = host=pg1-srv user=replic_user
restore_command = pgbackrest --stanza=demo archive-get %f "%p"
```

The `--type=standby` option creates the `standby.signal` needed for Postgres to start in standby mode. All we have to do now is to start the Postgres instances:

```
$ sudo systemctl enable edb-as-14.service
$ sudo systemctl start edb-as-14.service
```

If the replication setup is correct, you should see those processes on the **pg1-srv** server:

```
$ sudo -u enterprisedb ps -o pid,cmd fx
PID CMD
30594 /usr/edb/as14/bin/edb-postmaster -D /var/lib/edb/as14/data
...
31436 \_ postgres: walsender replic_user ... streaming 0/5001FB0
31439 \_ postgres: walsender replic_user ... streaming 0/5001FB0
30222 /usr/bin/pgbackrest server
```

We now have a 3-nodes cluster working with *Streaming Replication* and archives recovery as safety net.

8.8 Take backups from the standby servers

Add the following settings to the pgBackRest configuration file on **backup-srv** , in the [demo] stanza section:

```
pg2-host=pg2-srv
pg2-host-user=enterprisedb
pg2-path=/var/lib/edb/as14/data
pg2-host-type=tls
pg2-host-cert-file=certs/backup-srv.crt
pg2-host-key-file=certs/backup-srv.key
pg2-host-ca-file=certs/ca.crt
pg3-host=pg3-srv
pg3-host-user=enterprisedb
pg3-path=/var/lib/edb/as14/data
pg3-host-type=tls
pg3-host-cert-file=certs/backup-srv.crt
pg3-host-key-file=certs/backup-srv.key
pg3-host-ca-file=certs/ca.crt
backup-standby=y
```

Now, perform a backup fetching the data from the first standby server found in the configuration:

```
$ sudo -iu pgbackrest pgbackrest --stanza=demo --type=full backup
P00 INFO: backup command begin 2.38: ...
P00 INFO: execute non-exclusive pg_start_backup(): backup begins after the requested
        immediate checkpoint.
↪ completes
P00 INFO: backup start archive = 00000001000000000000000006, lsn = 0/6000028
P00 INFO: wait for replay on the standby to reach 0/6000028
P00 INFO: replay on the standby reached 0/6000028
P00 INFO: check archive for prior segment 000000010000000000000005
P00 INFO: execute non-exclusive pg_stop_backup() and wait for all WAL segments to
↪ archive
P00 INFO: backup stop archive = 00000001000000000000000006, lsn = 0/6000138
P00 INFO: check archive for segment(s)
↪ 00000001000000000000000006:00000001000000000000000006
P00 INFO: new backup label = 20220308-145104F
```

(continues on next page)

(continued from previous page)

```
P00 INFO: full backup size = 56.5MB, file total = 1691
P00 INFO: backup command end: completed successfully
```

8.9 Conclusion

The TLS server provides an alternative to SSH for remote operations. This new feature will fit perfectly in a containerized world where it can significantly improve performance and user experience.

‘APPENDIX’

9.1 Concepts

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

9.2 Backup

A backup is a consistent copy of a database cluster that can be restored to recover from a hardware failure, perform point-in-time recovery, or to bring up a new standby.

FullBackup : pgBackRest copies the entire content of the database cluster to the backup. The first backup of the database cluster is always a full backup. pgBackRest restores a full backup directly. The full backup does not depend on any files outside of the full backup for consistency.

DifferentialBackup : pgBackRest copies only the database cluster files that have changed since the last full backup. To restore a differential backup, pgBackRest copies 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 be valid to restore the differential backup.

IncrementalBackup : pgBackRest copies only the database cluster files that have changed since the last backup (which can be an incremental backup, a differential backup, or a full backup). Since an incremental backup only includes the files changed since the prior backup, they are generally much smaller than a full or differential backup. An incremental backup requires other backups to be valid before restoring the incremental backup. Since the incremental backup includes only those files that have been changed or added since the last backup, all prior incremental backups (back to the prior differential and the prior full backup) must be valid to restore the incremental backup. If no differential backup exists, then all prior incremental backups (back to the prior full backup and the full backup itself) must be valid to restore the incremental backup.

9.3 Restore

A restore is an 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.

9.4 Write Ahead Log (WAL)

WAL logging 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. Later, a background writer 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 eight hexadecimal digits represent the timeline and the next 16 digits are the log sequence number (LSN).

9.5 Encryption

Encryption is the process of converting data into a format that is unrecognizable unless you provide the appropriate password (also referred to as passphrase). To prevent unauthorized access to data stored within the repository, pgBackRest will encrypt the repository after authentication with a user-provided password.

9.6 Configuration Reference

You can find a complete list of commands and configuration keys in the project documentation:

- [Command reference](#) for command-line operations.
- [Configuration Reference](#) for creating pgBackRest configurations.

You can use pgBackRest entirely with command-line parameters, but a configuration file is more practical for complex installations when configuring multiple options instantly. The default location for the configuration file is `/etc/pgbackrest/pgbackrest.conf`. If the configuration file does not exist in that location, then the old `/etc/pgbackrest.conf` file will be loaded instead (if it exists).

It is also recommended to use a configuration file in order to avoid specifying secrets in command-line options.

9.7 Global

Use the global section of the configuration file to set the default values.

The global section also allows you to define values that will be used when invoking specific commands. For example, the following configuration setting would specify the compress level for the `archive-push` command:

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

9.8 Stanza

A **stanza** is a database cluster configuration that defines where the database cluster is located, how to take a backup of the database cluster, available archiving options, etc. Most database servers will only have one database cluster and, therefore, one stanza; whereas backup servers will have a stanza for each database cluster that needs to be backed up.

Because the stanza name will be used for the primary and all replicas, it is recommended to choose a name that describes the cluster's databases and actual function, such as `app` or `dw`, rather than the local instance name, such as `main` or `prod`.

The stanza-level settings overrides any global configuration settings.

9.9 Command Line Arguments and Environment Variables

You can also configure options as command line argument or environment variables.

- To use a non-boolean argument on the command line, simply prefix the option with a double dash.
- To set a boolean option to false on the command line, use the `--no-` prefix.
- To set an environment variable, capitalize the option name, prefix it with `PGBACKREST_`, and replace dashes with underscores.

The following table illustrates how to specify an option using the `pgbackrest.conf` file, command line argument, and environment variable.

Configuration File	Command Line Argument	Environment Variable
<code>pg1-host=1.2.3.4</code>	<code>-pg1-host=1.2.3.4</code>	<code>PGBACK- REST_PG1_HOST=1.2.3.4</code>
<code>compress=y</code>	<code>-compress</code>	<code>PGBACK- REST_COMPRESS=y</code>
<code>delta=n</code>	<code>-no-delta</code>	<code>PGBACKREST_DELTA=n</code>

9.10 Options Precedence

For all options, the order of precedence (highest to lowest) is as follows:

- Command line argument
- Environment variable
- `[stanza:command]`
- `[stanza]`
- `[global:command]`
- `[global]`
- Default (internal)

9.11 Configuring Multiple Database Hosts in a Stanza

Each stanza could configure multiple database hosts. As such, you can also configure every `pg1-*` configuration option for host 2 with `pg2-*`, for host 3 with `pg3-*`, etc. This assumes that those database clusters are linked together (e.g. using *Prepare the servers for Streaming Replication*).

‘FREQUENTLY ASKED QUESTIONS’

- Why do we have to specify the `--stanza` option even if only one stanza is defined in the configuration file?

Except for the repository specific commands (`info`, `start`, `stop`), the `--stanza` option is required so that adding a stanza does not cause the commands that did not have a stanza specified to suddenly throw errors because they do not know which stanza to use.

-
- Is it possible to only restore a specific database from a complete database cluster backup?

Yes. You can restore a specific database using the `restore -db-include <https://pgbackrest.org/command.html#command-restore/category-command/option-db-include>`__` option. After recovery, you can then extract the selected databases with the

`pg_dump` command.

The databases that are not explicitly included with this option will be restored as sparse, zeroed files to save space; however, it still allows PostgreSQL to perform the recovery using less disk space. You can then remove these databases with the `DROP DATABASE` command since they will no longer be accessible.

-
- Is it possible to setup pgBackRest to access remote hosts without using the SSH protocol?

Yes. Since the 2.37 release, pgBackRest can use TLS with client certificates to enable communication between the hosts. A TLS server must then be configured and started on each host.

See [Setup TLS](<https://pgbackrest.org/user-guide-rhel.html#repo-host/config>) for more information.

-
- How many CPU cores are needed on a dedicated remote backup server?

Since compression/decompression is performed on the Postgres host, the repository host does only need to encrypt/decrypt SSH or TLS connections. Usually 1 repository core per 4 Postgres cores is a good starting point.