pg_probackup 3.2.1 Documentation

https://postgrespro.com

pg_probackup 3.2.1 Documentation

Chapter 1. About pg_probackup3

pg_probackup3 is a solution to manage local or remote backup and recovery of Postgres Pro database clusters. It is designed to perform backups of the Postgres Pro instance that enable you to restore the server in case of a failure. It supports Postgres Pro and PostgreSQL 14 or higher.

pg_probackup3 includes all the key functionalities of the prior versions of the pg_probackup utility. Some less popular features may be missing at the moment, but will be implemented in the future.

1.1. Key Features

pg_probackup3 allows you to take full or incremental backups and provides all necessary features for backup management:

- FULL backups. They contain all the data files required to restore the database cluster.
- Incremental backups. They operate at the page level, only storing the data that has changed since the previous backup. It allows you to save disk space and speed up the backup process as compared to taking full backups. It is also faster to restore the cluster by applying incremental backups than by replaying WAL files. pg_probackup3 supports the following modes of incremental backups:
  - DELTA backup. In this mode, pg_probackup3 reads all data files in the data directory and copies only those pages that have changed since the previous backup. This mode can create read-only I/O load equal to that of a full backup.
  - PTRACK backup. In this mode, Postgres Pro tracks page changes on the fly. Continuous archiving is not necessary for it to operate. Each time a relation page is updated, this page is marked in a special PTRACK bitmap. Tracking implies some minor overhead on the database server operation, but speeds up incremental backups significantly.
- Guaranteed data integrity in backups: pg_probackup3 can take only physical online backups, and online backups require WAL for consistent recovery. So regardless of the chosen backup mode (FULL or DELTA), any backup taken with pg_probackup3 must use one of the following *WAL delivery modes*:
  - STREAM. Such backups include all the files required to restore the cluster to a consistent state at the time the backup was taken. Regardless of continuous archiving having been set up or not, the WAL segments required for consistent recovery are streamed via the replication protocol during backup and included into the backup files. That's why such backups are called *autonomous*, or *standalone*.
  - ARCHIVE. Such backups rely on continuous archiving to ensure consistent recovery. This is the default WAL delivery mode.
- To manage backup data, pg_probackup3 creates a *backup catalog*. This is a directory that stores all backup files with additional meta information, as well as WAL archives required for point-in-time recovery. The list of backups stored in the catalog as well as the list of all WAL timelines and the corresponding meta information can be derived in plain text or in the JSON format.
- You can store backups for different instances in separate subdirectories of a single backup catalog. pg_probackup3 allows storing backups both on local disks and in network file systems. It provides support for working through the following network storage protocols:
  - NFS version 3 and 4
  - S3 based on MinIO object, Amazon S3, and VK Cloud storage. Backup data is transferred to and from S3 without saving it in intermediate locations thus eliminating the need of having a large temporary storage.
  - WebDav
  - SAMBA
  - FTP/SFTP
- Retention: Managing WAL archive and backups in accordance with retention policy. You can configure retention policy based on recovery time or the number of backups to keep, as well as specify time to live (TTL) for a particular backup. Expired backups can be merged or deleted.
- Multithreading: It is possible to run `backup`, `restore`, `merge`, `delete`, `catchup`, and `validate` processes on multiple parallel threads.

- Remote operations: Backing up Postgres Pro instance located on a remote system or restoring a backup remotely.
- External directories: Backing up files and directories located outside of the Postgres Pro data directory (`PGDATA`), such as scripts, configuration files, logs, or SQL dump files.
- Partial restore:
  - Restoring only the specified databases
  - Point-in-time recovery (PITR)

1.2. New Features of pg_probackup3

As compared to *pg_probackup*, pg_probackup3 comprises the following new features and improvements:

- Version independence: The same pg_probackup3 version can now be used with different versions of Postgres Pro or PostgreSQL, ensuring compatibility and flexibility.
- API integration: pg_probackup3 provides programming interfaces for integration, which enables external backup management systems to utilize pg_probackup3 functionality for creating Postgres Pro backups and ensuring centralized backup management.
- Work without SSH: pg_probackup3 can work without an SSH connection, simplifying data transfer.
- FUSE: pg_probackup3 introduces the new `fuse` command, which enables running a database instance directly from a backup without requiring a full restore, using the FUSE (Filesystem in User Space) mechanism.
- Operation by unprivileged users: pg_probackup3 can be started by users who do not have access rights to PGDATA. This helps to increase security and reduce the risk of potential errors.
- A new backup format: Each backup is now stored as a single file, simplifying backup management and storage while providing additional usage capabilities.
- pg_probackup3 supports various backup protocols via the following backup data source modes:
  - BASE: Leverages pg_basebackup replication protocol.
  - PRO: Uses the pg_probackup3's proprietary replication protocol, ensuring secure, high-speed data transfer from the Postgres Pro server.
  - DIRECT: Bypasses replication protocols entirely, accessing data files directly for backup operations.

  For details on backup data source modes, see the next section.
- Merging incremental backup chains: It is now possible to merge chains of incremental backups, thus saving storage space and enabling more flexible backup policies.
- Tape-ready: pg_probackup3's new backup format eliminates file fragmentation and allows size customization, thus providing optimal performance with tape backup systems.
- Completely reengineered core
- Redesigned architecture
- Improved performance

1.3. pg_probackup3 Backup Data Source Modes

The pg_probackup3 utility supports the following backup data source modes: BASE, DIRECT, and PRO. The chosen mode determines how the connection to the database server is established and how the files for creating the backup are obtained.

The table below provides a concise overview of the key differences between these backup data source modes.

Backup Data Source Mode	**Data Transfer Method**	**libpg-probackup Library Required**	**Validation**	**CFS Support**	**Additional Extensions Required**
BASE	Via the pg_basebackup replication protocol	No	Yes	No	None

Backup Data Source Mode	Data Transfer Method	libpgprobackup Library Required	Validation	CFS Support	Additional Extensions Required
DIRECT	Direct filesystem access	No	Yes	Yes	PTRACK (for incremental backups)
PRO	Via the proprietary replication protocol	Yes	Yes	Yes	pgpro_bindump (for protocol implementation), PTRACK (for incremental backups)

A detailed description of each mode is provided in the sections below.

1.3.1. BASE

The BASE mode uses the standard pg_basebackup replication protocol. Data is copied without using advanced change-tracking and segmentation mechanisms.

Features:

- Copies data via the replication protocol using the `libpq` library.
- Compatible with PostgreSQL and requires no additional extensions.

Limitations:

- Requires access to system functions.
- Does not support accelerated incremental mechanisms (PTRACK).
- Generally slower for large data volumes compared to other modes.

1.3.2. DIRECT

In the DIRECT mode, the pg_probackup3 utility gains direct access to the `PGDATA` directory via the filesystem, bypassing the replication protocol for file transfer.

Features:

- Uses a standard database connection.
- Performs data integrity checks during transfer.
- Suitable for cases where access via the replication protocol is unavailable or undesirable.

Limitations:

- Requires direct access to the server filesystem.
- Requires access to system functions.
- For remote servers, requires SSH-based filesystem access. SSH is not required for accessing `PGDATA`.

1.3.3. PRO

The PRO mode uses a dedicated library, which includes data processing logic, and its own backup protocol. This protocol is specifically designed for secure and fast data transfer from the server.

Features:

- Connects to the database via the system `libpq` library.
- Transfers data via its own proprietary replication protocol.
- Supports all backup types: FULL, DELTA, and PTRACK.

- Optimized for performance and reduced server load.
- Performs data integrity checks during transfer.

Limitations:

- Requires the pgpro_bindump plugin to be installed on the server side.
- Requires the `libpb3_encoder.so` library to be installed on the server. No separate library is required for pg_probackup3.
- Can only be used with Postgres Pro.
- Configuring continuous WAL archiving to a remote server requires an SSH connection (not required for STREAM mode).

1.4. pg_probackup3 Limitations

pg_probackup3 currently has the following limitations:

- Any FULL or DELTA backup taken with pg_probackup3 must use one of these WAL delivery modes: ARCHIVE or STREAM.
- The remote mode is not supported on Windows systems.
- The Postgres Pro server from which the backup was taken and the restored server must be compatible by the *block_size* and *wal_block_size* parameters and have the same major release number. Depending on cluster configuration, Postgres Pro itself may apply additional restrictions, such as CPU architecture or libc/icu versions.
- pg_probackup3 only supports Postgres Pro and PostgreSQL 14 or higher.
- For Postgres Pro and PostgreSQL 14: If a new database is created in the instance between a FULL and DELTA backup, incremental backups will fail until a new FULL backup is taken.
- Creating backups using different backup data source modes (`--backup-source`) within one backup chain is disabled.
- The ability to run pg_probackup3 on parallel threads (using the `-j` option) is currently implemented only for the following commands: `backup`, `restore`, `merge`, `catchup`, and `validate`.
- The default path for WAL files during backup and restore is set to `PGDATA/pg_wal`.
- The `--dry-run` option only works for the `delete` command.
- Validation is only available for the whole backup.
- Incremental backups are not supported during TDE status transitions (enabling or disabling). Following any TDE status change, a full backup must be performed.
- `merge` operations are prohibited if incremental backup chains contain a backup capturing a TDE status change. In such cases, a complete cluster-level full backup is required.
- Clusters with TDE enabled do not support `catchup` or `fuse` operations.

Chapter 2. Installation and Setup

2.1. Installing pg_probackup3

To install pg_probackup3, follow the steps below.

1. Add the pg_probackup3 package repository to your operating system. You can get the exact repositories and commands for supported Linux distributions from the Postgres Pro support team.
2. Install the packages.

   On Debian family systems:

   ```
   sudo apt update
   sudo apt install pg-probackup3
   ```

   On Red Hat family systems:

   ```
   sudo dnf install pg-probackup3
   ```

   You may need to use `yum` instead of `dnf` on older systems.
3. Print the pg_probackup3 version to verify the installation:

   ```
   pg_probackup3 --version
   ```

4. To use pg_probackup3 in the PRO mode, configure Postgres Pro as follows:

   1. Install the *pgpro_bindump* plugin.
   2. Add the following parameters to `postgresql.conf`:

      ```
      shared_preload_libraries = 'pgpro_bindump'
      wal_level = 'replica'  # or 'logical'
      walsender_plugin_libraries = 'pgpro_bindump'
      ```

   3. Restart the Postgres Pro instance.

2.2. Initializing a Backup Catalog

pg_probackup3 stores all WAL and backup files in the corresponding subdirectories of a backup catalog.

To initialize a backup catalog, run the following command:

```
pg_probackup3 init -B backup_dir
```

where *backup_dir* is the path to the backup catalog. If *backup_dir* already exists, it must be empty. Otherwise, pg_probackup3 returns an error.

pg_probackup3 creates the *backup_dir* backup catalog, with the following subdirectories:

- `wal/` — directory for WAL files.
- `backups/` — directory for backup files.

Once a backup catalog is initialized, you can add a new backup instance.

2.3. Adding a New Backup Instance

pg_probackup3 can store backups for multiple database clusters in a single backup catalog. To set up the required subdirectories, you must add a backup representation (instance) of each database cluster you are going to back up to the backup catalog.

Before adding a new backup instance, ensure that pg_probackup3 can connect to the database cluster you are going to back up. To add a new backup instance, run the following command:

```
pg_probackup3 add-instance -B backup_dir -D data_dir --instance=instance_name
 [ssh_options]
```

Where:

- *data_dir* is the data directory of the cluster you are going to back up. To set up and use pg_probackup3, write access to this directory is required.
- *instance_name* is the name of the subdirectories that will store WAL and backup files for this cluster.
- *ssh_options* are optional parameters that need to be specified only if *data_dir* is located on a remote system.

pg_probackup3 creates the *instance_name* subdirectories under the `backups/` and `wal/` directories of the backup catalog. The `backups/`*instance_name* directory contains the `pg_probackup3.conf` configuration file that controls pg_probackup3 settings for this backup instance. If you run this command with the SSH options, the specified parameters will be added to `pg_probackup3.conf`.

For details on how to fine-tune pg_probackup3 configuration, see Section 2.4.

The user running pg_probackup3 must have full access to *backup_dir* directory and at least read-only access to *data_dir* directory. If you specify the path to the backup catalog in the `BACKUP_PATH` environment variable, you can omit the corresponding option when running pg_probackup3 commands.

> **Note**
>
> It is recommended to use the *--allow-group-access* option, so that backups can be done by any OS user in the same group as the cluster owner. In this case, the user should have read permissions for the cluster directory.

2.4. Configuring pg_probackup3

Once a backup catalog is initialized and a new backup instance is added, you can use the `pg_probackup3.conf` configuration file located in the *backup_dir*`/backups/`*instance_name* directory to fine-tune pg_probackup3 configuration of this instance.

For example, the backup command uses a regular Postgres Pro connection. To avoid specifying connection options each time on the command line, you can set them in the `pg_probackup3.conf` configuration file using the set-config command.

Initially, `pg_probackup3.conf` contains the following settings:

- `PGDATA` — the path to the data directory of the cluster to back up.
- `system-identifier` — the unique identifier of the Postgres Pro instance.

Additionally, you can define retention, logging, and compression settings using the `set-config` command:

```
pg_probackup3 set-config -B backup_dir --instance=instance_name
[--external-dirs=external_directory_path] [connection_options] [compression_options]
 [retention_options] [logging_options]
```

To view the current settings, run the following command:

```
pg_probackup3 show-config -B backup_dir --instance=instance_name
```

You can override the settings defined in `pg_probackup3.conf` when running pg_probackup3 commands via the corresponding environment variables and/or command line options.

> **Note**
>
> It is **not recommended** to edit `pg_probackup3.conf` manually. Modifying the configuration file using the set-config command prevents accidental typographical errors.

2.5. Specifying Connection Settings

If you define connection settings in the `pg_probackup3.conf` configuration file, you can omit connection options in all the subsequent pg_probackup3 commands. However, if the corresponding environment variables are set, they get higher priority. The options provided on the command line overwrite both environment variables and configuration file settings.

If nothing is given, the default values are taken. By default, pg_probackup3 tries to use local connection via Unix domain socket (`localhost` on Windows) and tries to get the database name and the user name from the `PGUSER` environment variable or the current OS user name.

2.6. Configuring the Database Cluster

Although pg_probackup3 can be used by a superuser, it is recommended to create a separate role with the minimum permissions required for the chosen backup strategy. In these configuration instructions, the `backup` role is used as an example.

For security reasons, it is recommended to run the configuration SQL queries below in a separate database.

```
postgres=# CREATE DATABASE backupdb;
postgres=# \c backupdb
```

To perform a backup, the following permissions for role `backup` are required only in the database **used for connection** to the Postgres Pro server:

```
BEGIN;
CREATE ROLE backup WITH LOGIN REPLICATION;
GRANT USAGE ON SCHEMA pg_catalog TO backup;
GRANT SELECT ON TABLE pg_catalog.pg_tablespace TO backup;
GRANT EXECUTE ON FUNCTION pg_catalog.current_setting(text) TO backup;
GRANT EXECUTE ON FUNCTION pg_catalog.set_config(text, text, boolean) TO backup;
GRANT EXECUTE ON FUNCTION pg_catalog.pg_is_in_recovery() TO backup;
GRANT EXECUTE ON FUNCTION pg_catalog.pg_backup_start(text, boolean) TO backup;
GRANT EXECUTE ON FUNCTION pg_catalog.pg_backup_stop(boolean) TO backup;
GRANT EXECUTE ON FUNCTION pg_catalog.pg_create_restore_point(text) TO backup;
GRANT EXECUTE ON FUNCTION pg_catalog.pg_switch_wal() TO backup;
GRANT EXECUTE ON FUNCTION pg_catalog.pg_last_wal_replay_lsn() TO backup;
GRANT EXECUTE ON FUNCTION pg_catalog.txid_current() TO backup;
GRANT EXECUTE ON FUNCTION pg_catalog.txid_current_snapshot() TO backup;
GRANT EXECUTE ON FUNCTION pg_catalog.txid_snapshot_xmax(txid_snapshot) TO backup;
GRANT EXECUTE ON FUNCTION pg_catalog.pg_control_checkpoint() TO backup;
COMMIT;
```

In the *pg_hba.conf* file, allow connection to the database cluster on behalf of the `backup` role.

> **Note**
>
> Direct access to `PGDATA` is not needed for backup creation in the PRO or BASE modes, but is required in the DIRECT mode. The PRO mode is set by default.

Depending on whether you plan to take standalone or archive backups, Postgres Pro cluster configuration will differ, as specified in the sections below. To run pg_probackup3 in the remote mode or create PTRACK backups, additional setup is required.

For details, see the sections Setting up STREAM Backups, Setting up continuous WAL archiving, Configuring SSH Connection, and Setting up PTRACK Backups.

2.7. Setting up STREAM Backups

To set up the cluster for STREAM backups, complete the following steps:

- If the `backup` role does not exist, create it with the `REPLICATION` privilege when Configuring the Database Cluster:

```
CREATE ROLE backup WITH LOGIN REPLICATION;
```

- If the `backup` role already exists, grant it with the `REPLICATION` privilege:

```
ALTER ROLE backup WITH REPLICATION;
```

- In the *pg_hba.conf* file, allow replication on behalf of the `backup` role.
- Make sure the parameter *max_wal_senders* is set high enough to leave at least one session available for the backup process.
- Set the parameter *wal_level* to be higher than `minimal`.

If you are planning to perform PITR with STREAM backups, you still have to configure WAL archiving, as explained in the section Setting up continuous WAL archiving.

Once these steps are complete, you can start taking FULL, DELTA, and PTRACK backups in the STREAM WAL mode.

> **Note**
>
> If you are planning to rely on *.pgpass* for authentication when running backup in STREAM mode, then .pgpass must contain credentials for `replication` database, used to establish connection via replication protocol. Example: pghost:5432:replication:backup_user:my_strong_password

2.8. Setting up Continuous WAL Archiving

Performing PITR and making backups with the ARCHIVE WAL delivery mode require *continuous WAL archiving* to be enabled. To set up continuous archiving in the cluster, complete the following steps:

- Make sure the *wal_level* parameter is higher than `minimal`.
- If you are configuring archiving on the primary, *archive_mode* must be set to `on` or `always`.
- Set the *archive_command* parameter, as follows:

```
archive_command = '"install_dir/pg_probackup3" archive-push -B "backup_dir" --
instance=instance_name --wal-file-name=%f [ssh_options]'
```

where *install_dir* is the installation directory of the pg_probackup3 version you are going to use, *backup_dir* and *instance_name* refer to the already initialized backup catalog instance for this database cluster, and SSH options only need to be specified to archive WAL on a remote host. For details about all possible `archive-push` parameters, see the section archive-push.

Once these steps are complete, you can start making backups in the ARCHIVE WAL mode as well as perform PITR.

You can view the current state of the WAL archive using the show command. For details, see Section 3.3.2.

> **Note**
>
> Instead of using the archive-push command provided by pg_probackup3, you can use any other tool to set up continuous archiving as long as it delivers WAL segments into *backup_dir*/wal/*in-*

stance_name directory. If compression is used, it should be `gzip`, and `.gz` suffix in filename is mandatory.

Note

Instead of configuring continuous archiving by setting the `archive_mode` and `archive_command` parameters, you can opt for using the *pg_receivewal* utility. In this case, pg_receivewal `-D` *directory* option should point to *backup_dir*`/wal/`*instance_name* directory. pg_probackup3 supports WAL compression that can be done by pg_receivewal. “Zero Data Loss” archive strategy can be achieved only by using pg_receivewal.

2.9. Setting up Partial Restore

If you are planning to use partial restore with the `--db-exclude-name` or `--db-include-name` options, complete the following additional step:

- Grant the read-only access to `pg_catalog.pg_database` to the `backup` role only in the database **used for connection** to Postgres Pro server:

```
GRANT SELECT ON TABLE pg_catalog.pg_database TO backup;
```

2.10. Setting up PTRACK Backups

The PTRACK backup mode can be used only for Postgres Pro Standard and Postgres Pro Enterprise installations, or patched vanilla PostgreSQL.

If you are going to use PTRACK backups, complete the steps below.

Note

The permissions required for the role that will perform PTRACK backups (the `backup` role in the examples below) are listed in Section 2.6. The role must have permissions only in the database used for connection to the Postgres Pro server.

1. Add `ptrack` to the `shared_preload_libraries` variable in the `postgresql.conf` file:

   ```
   shared_preload_libraries = 'ptrack'
   ```

2. To enable tracking page updates, set the `ptrack.map_size` parameter to a positive integer and restart the server.

   For optimal performance, it is recommended to set `ptrack.map_size` to *N* `/ 1024`, where *N* is the size of the Postgres Pro cluster, in MB. If you set this parameter to a lower value, PTRACK is more likely to map several blocks together, which leads to false-positive results when tracking changed blocks and increases the incremental backup size as unchanged blocks can also be copied into the incremental backup. Setting `ptrack.map_size` to a higher value does not affect PTRACK operation, but it is not recommended to set this parameter to a value higher than 1024.

   Note

   If you change the `ptrack.map_size` parameter value, the previously created PTRACK map file is cleared, and tracking newly changed blocks starts from scratch. Thus, you have to retake a full backup before taking incremental PTRACK backups after changing `ptrack.map_size`.

3. Create PTRACK extension:

```
CREATE EXTENSION ptrack;
```

2.11. Configuring SSH Connection

pg_probackup3 supports the remote mode that allows you to perform backup and WAL archiving operations remotely. To operate in the remote mode, pg_probackup3 uses an SSH connection between the Postgres Pro server (where pg_probackup3 is run) and the backup storage server. This allows managing backups on a remote host as if they were stored locally.

pg_probackup3 can store and read backup files and metadata on an SSH server using the SFTP protocol. This operation scheme is similar to that of S3.

If you are going to use pg_probackup3 in the remote mode via SSH, set up a passwordless SSH connection to the server via a public and private keys: set the public key on the server side and the private one — on the client.

See the SSH Options section for details on the remote connection parameters.

pg_probackup3 in the remote mode via SSH works as follows:

- All commands can be launched in the remote mode.
- Operating in the remote mode does not require the pg_probackup3 binary to be installed on the remote system.

You can provide SSH server settings in the configuration file using the `--config-file` option. See the section Common Options for details.

2.12. Configuring S3 Connection

pg_probackup3 supports S3 interface for storing backups. Backup data is transferred to and from S3 without saving it in intermediate locations thus eliminating the need of having a large temporary storage.

> **Note**
>
> S3 is available only when pg_probackup3 is used with Postgres Pro Enterprise.

If you are going to use pg_probackup3 with S3 interface, complete the following steps:

- Create a bucket with a unique and meaningful name in the S3 storage for you future backups.
- Create ACCESS_KEY and SECRET_ACCESS_KEY tokens to be used for secure connectivity instead of your username and password.
- For communication between pg_probackup3 and S3 server, set values of environment variables corresponding to your S3 server. For example:

  ```
  export PG_PROBACKUP_S3_HOST=127.0.0.1
  export PG_PROBACKUP_S3_PORT=9000
  export PG_PROBACKUP_S3_REGION=ru-msk
  export PG_PROBACKUP_S3_BUCKET_NAME=test1
  export PG_PROBACKUP_S3_ACCESS_KEY=admin
  export PG_PROBACKUP_S3_SECRET_ACCESS_KEY=password
  export PG_PROBACKUP_S3_HTTPS=ON
  ```

  Alternatively, you can provide S3 server settings in the configuration file or by using the command-line options. For more details, refer to the `--config-file` option in the section Common Options and to the section S3 Options.

  It makes sense to specify S3 server settings if `--s3=minio`, as described in the section S3 Options.

  The following environment variables can be specified:

`PG_PROBACKUP_S3_HOST`

Address of the S3 server. Can include the port number, separated by a colon. If the port number is not specified in a host string, the value of `PG_PROBACKUP_S3_PORT` is assumed. Do not add a colon if the port number is not specified.

For example:

```
export PG_PROBACKUP_S3_PORT=80
export PG_PROBACKUP_S3_HOST="127.0.0.1:9000"
```

In this example, for the “127.0.0.1” address, the port 9000 is explicitly specified and will be used instead of value 80, specified through `PG_PROBACKUP_S3_PORT`.

`PG_PROBACKUP_S3_PORT`

The port of the S3 server.

`PG_PROBACKUP_S3_REGION`

The region of the S3 server. The default value is `us-east-1`.

`PG_PROBACKUP_S3_BUCKET_NAME`

The name of the bucket on the S3 server.

`PG_PROBACKUP_S3_ACCESS_KEY`
`PG_PROBACKUP_S3_SECRET_ACCESS_KEY`

Secure tokens on the S3 server.

`PG_PROBACKUP_S3_HTTPS`

The protocol to be used. Possible values:

- `ON` or `HTTPS` — use HTTPS
- Other than `ON` or `HTTPS` — use HTTP

`PG_PROBACKUP_S3_BUFFER_SIZE`

The size of the read/write buffer for communicating with S3, in MiB. The default is `16`.

`PG_PROBACKUP_S3_RETRIES`

The maximum number of attempts to execute an S3 request in case of failures. The default is `3`.

`PG_PROBACKUP_S3_TIMEOUT`

The maximum amount of time to execute an HTTP request to the S3 server, in seconds. The default is `300`.

`PG_PROBACKUP_S3_IGNORE_CERT_VER`

Don't verify the certificate host and peer. The default is `OFF`.

`PG_PROBACKUP_S3_CA_CERTIFICATE`

Specify the path to file with trust Certificate Authority (CA) bundle.

`PG_PROBACKUP_S3_CA_PATH`

Specify the directory with trust CA certificates.

`PG_PROBACKUP_S3_CLIENT_CERT`

Setup SSL client certificate.

`PG_PROBACKUP_S3_CLIENT_KEY`

Setup private key file for TLS and SSL client certificate.

2.12.1. S3 Required Permissions

The following minimal permissions to the target S3 bucket should be granted for the access key used by pg_probackup3 with versioning disabled:

- for the init command:

```
s3:GetBucketVersioning
s3:ListBucket
```

- for the add-instance, set-config, and set-backup commands:

```
s3:GetBucketVersioning
s3:ListBucket
s3:PutObject
```

- for the del-instance command:

```
s3:GetBucketVersioning
s3:ListBucket
s3:DeleteObject
```

- for the backup, archive-push, and file-map commands:

```
s3:ListBucket
s3:PutObject
s3:GetBucketVersioning
s3:AbortMultipartUpload
s3:GetObject
```

- for the restore, fuse, show, show-config, and send-backup commands:

```
s3:GetBucketVersioning
s3:GetObject
s3:ListBucket
```

- for the validate command:

```
s3:GetBucketVersioning
s3:GetObject
s3:ListBucket
s3:PutObject
```

- for the merge and retention commands:

```
s3:ListBucket
s3:PutObject
s3:GetBucketVersioning
s3:AbortMultipartUpload
s3:GetObject
s3:DeleteObject
```

- for the delete command:

```
s3:GetBucketVersioning
s3:GetObject
s3:ListBucket
s3:DeleteObject
```

When versioning is enabled, the following additional permissions are required:

- `s3:ListBucketVersions` for all commands that require `s3:ListBucket`
- `s3:DeleteObjectVersion` for all commands that require `s3:DeleteObject`

Chapter 3. Use Cases

3.1. Creating a Backup

To create a backup, run the following command:

```
pg_probackup3 backup -B backup_dir --instance=instance_name -b backup_mode -
s backup_source -i backup_id
```

Where *backup_mode* can take one of the following values: FULL, DELTA, and PTRACK.

And *backup_source* can take one of these: DIRECT, BASE, and PRO.

> **Warning**
>
> The BASE and DIRECT backup data source modes do not support CFS.

> **Note**
>
> The BASE backup data source mode supports only FULL and DELTA backup modes.

Some options can be skipped depending on the user goals:

- If *backup_mode* is not specified, the FULL mode is used by default.
- PRO is the default value for *backup_source*.
- If the backup ID is not specified explicitly in the body of a request, *backup_id* will take the value of the date and time it was created.
- If the backup ID is specified and includes a path to a directory, *backup_dir* and *instance_name* can be skipped without specification. Example: -i /mnt/ramdisk/backups/2.backup.
- If a path to the data directory is not specified either via *backup_dir* or via --backup-id, the current directory will be used as the default one.
- If you omit the **parent** backup ID when performing incremental backups, pg_probackup3 will use the latest valid backup from the backup chain. If pg_probackup3 somehow fails to find it, the backup process will conclude with an error.
- If the --from-full parameter is specified, an incremental backup will be created from the last FULL backup.

3.1.1. ARCHIVE Mode

ARCHIVE is the default WAL delivery mode.

To make a FULL backup in the ARCHIVE mode, run:

```
pg_probackup3 backup -B backup_dir --instance=instance_name -b FULL
```

ARCHIVE backups rely on continuous archiving to get WAL segments required to restore the cluster to a consistent state at the time the backup was taken.

3.1.2. STREAM Mode

To make a FULL backup in the STREAM mode, add the --stream flag to the command from the previous example:

```
pg_probackup3 backup -B backup_dir --instance=instance_name -b FULL --stream
```

Unlike backups in the ARCHIVE mode, STREAM backups include all the WAL segments required to restore the cluster to a consistent state at the time the backup was taken.

During backup pg_probackup3 streams WAL files containing WAL records between `Start LSN` and `Stop LSN` to the backup file.

Even if you are using continuous archiving, STREAM backups can still be useful in the following cases:

- STREAM backups can be restored on the server that has no file access to WAL archive.
- STREAM backups enable you to restore the cluster state at the point in time for which WAL files in archive are no longer available.

3.1.3. External Directories

To back up a directory located outside of the data directory, use the optional `--external-dirs` parameter that specifies the path to this directory. If you would like to add more than one external directory, you can provide several paths separated by colons on Linux systems.

For example, to include `/etc/dir1` and `/etc/dir2` directories into the full backup of your *`instance_name`* instance that will be stored under the *`backup_dir`* directory on Linux, run:

```
pg_probackup3 backup -B backup_dir --instance=instance_name -b FULL --external-dirs=/
etc/dir1:/etc/dir2
```

Similarly, to include `C:\dir1` and `C:\dir2` directories into the full backup on Windows, run:

```
pg_probackup3 backup -B backup_dir --instance=instance_name -b FULL --external-dirs=C:
\dir1;C:\dir2
```

pg_probackup3 recursively copies the contents of each external directory into a separate subdirectory in the backup catalog. Since external directories included into different backups do not have to be the same, when you are restoring the cluster from an incremental backup, only those directories that belong to this particular backup will be restored. Any external directories stored in the previous backups will be ignored.

To include the same directories into each backup of your instance, you can specify them in the `pg_probackup3.conf` configuration file using the set-config command with the `--external-dirs` option.

> **Note**
>
> External directories are not supported in the BASE mode.

3.2. Restoring a Cluster

3.2.1. Full Restore

To restore the database cluster from a backup, run the restore command with at least the following options:

```
pg_probackup3 restore -B backup_dir --instance=instance_name -i backup_id
```

Where:

- *`backup_dir`* is the backup catalog that stores all backup files and meta information.
- *`instance_name`* is the backup instance for the cluster to be restored.
- *`backup_id`* specifies the backup to restore the cluster from.

If you restore ARCHIVE backups or perform PITR, pg_probackup3 creates a recovery configuration file once all data files are copied into the target directory. This file includes the minimal settings required for recovery, except for the password in the *primary_conninfo* parameter; you have to add the password

manually or use the `--primary-conninfo` option, if required. pg_probackup3 writes recovery settings into the `probackup_recovery.conf` file and then includes it into `postgresql.auto.conf`.

If you are restoring a STREAM backup, the restore is complete at once, with the cluster returned to a self-consistent state at the point when the backup was taken. For ARCHIVE backups, Postgres Pro replays all available archived WAL segments, so the cluster is restored to the latest state possible within the current timeline. You can change this behavior by using the recovery target options with the `restore` command, as explained in Section 3.2.4.

If the cluster to restore contains tablespaces, pg_probackup3 restores them to their original location by default. To restore tablespaces to a different location, use the `--tablespace-mapping`/`-T` option. Otherwise, restoring the cluster on the same host will fail if tablespaces are in use, because the backup would have to be written to the same directories.

When using the `--tablespace-mapping`/`-T` option, you must provide absolute paths to the old and new tablespace directories. If a path happens to contain an equals sign (=), escape it with a backslash. This option can be specified multiple times for multiple tablespaces. For example:

```
pg_probackup3 restore -B backup_dir --instance=instance_name -D data_dir -j 4 -
i backup_id -T tablespace1_dir=tablespace1_newdir -T tablespace2_dir=tablespace2_newdir
```

3.2.2. Incremental Restore

The speed of restore from backup can be significantly improved by replacing only invalid and changed pages in already existing Postgres Pro data directory using incremental restore options with the restore command.

> **Note**
>
> Currently, incremental restore is only available in the PRO mode. Support for BASE and DIRECT modes will be implemented in future releases.

To restore the database cluster from a backup in incremental mode, run the restore command with the following options:

```
pg_probackup3 restore -B backup_dir --instance=instance_name -D data_dir -
I incremental_mode
```

Where *`incremental_mode`* can take one of the following values:

- CHECKSUM — read all data files in the data directory, validate header and checksum in every page and replace only invalid pages and those with checksum and LSN not matching with corresponding page in backup. This is the simplest, the most fool-proof incremental mode. Recommended to use by default.
- LSN — read the `pg_control` in the data directory to obtain redo LSN and redo TLI, which allows you to determine a point in history(shiftpoint), where data directory state shifted from target backup chain history. If shiftpoint is not within reach of backup chain history, then restore is aborted. If shiftpoint is within reach of backup chain history, then read all data files in the data directory, validate header and checksum in every page and replace only invalid pages and those with LSN greater than shiftpoint. This mode offers a greater speed up compared to CHECKSUM, but rely on two conditions to be met. First, the *data checksums* parameter must be enabled in the data directory (to avoid corruption due to hint bits). This condition will be checked at the start of incremental restore and the operation will be aborted if checksums are disabled. Second, the `pg_control` file must be synchronized with the state of the data directory. This condition cannot be checked at the start of the restore, so it is the user's responsibility to ensure that `pg_control` contains valid information. Therefore, it is not recommended to use LSN mode in any situation where `pg_control` cannot be trusted or has been tampered with: for example, after `pg_resetxlog` execution or after a restore from backup without recovery being run.
- NONE — regular restore without any incremental optimizations.

Regardless of the chosen incremental mode, pg_probackup3 will check that `postmaster` in given destination directory is not running and `system-identifier` is the same as in the backup.

An example of using incremental restore with a tablespace in CHECKSUM mode:

```
========================================================================================================
Instance Version ID      End time                 Mode  WAL Mode TLI Duration Data WAL
 Zalg Zratio Start LSN  Stop LSN   Status
========================================================================================================
inst     17      1-full  2025-08-05 16:14:10+0700 FULL  STREAM   1   1s       47MB -
 none 1.46   0/A3000028 0/A3000160 OK
inst     17      1-delta 2025-08-05 16:14:33+0700 DELTA STREAM   1   1s       21MB -
 none 3.18   0/A5000028 0/A5000160 OK
inst     17      2-delta 2025-08-05 16:14:37+0700 DELTA STREAM   1   0ms      21MB -
 none 3.18   0/A6000028 0/A6000160 OK

backup_user@backup_host:~$ ./pg_probackup3 restore -D /mnt/node -B /mnt/b3b --
instance=inst -I checksum --backup-id 2-delta -T /mnt/ts1=/mnt/ts2 --threads=1
[2025-08-05 16:20:38.061117] [264669] [0x79693fc1a4c0] [info] command: ./pg_probackup3
 restore -D /mnt/node -B /mnt/backups --instance=inst -I checksum --backup-id 2-delta -
T /mnt/ts1=/mnt/ts2 --threads=1
[2025-08-05 16:20:38.061244] [264669] [0x79693fc1a4c0] [info] execute command:
 'restore', instance 'inst', backup_id '2-delta'
[2025-08-05 16:20:38.061855] [264669] [0x79693fc1a4c0] [info] Start validate 2-
delta ...
[2025-08-05 16:20:38.065368] [264669] [0x79693d5a66c0] [info] Validating backup 1-full
 chunk #0 out of 8
[2025-08-05 16:20:38.088840] [264669] [0x79693d5a66c0] [info] Validating backup 1-full
 chunk #1 out of 8
[2025-08-05 16:20:38.112253] [264669] [0x79693d5a66c0] [info] Validating backup 1-full
 chunk #2 out of 8
[2025-08-05 16:20:38.193395] [264669] [0x79693d5a66c0] [info] Validating backup 1-full
 chunk #3 out of 8
[2025-08-05 16:20:38.213982] [264669] [0x79693d5a66c0] [info] Validating backup 1-full
 chunk #4 out of 8
[2025-08-05 16:20:38.235532] [264669] [0x79693d5a66c0] [info] Validating backup 1-full
 chunk #5 out of 8
[2025-08-05 16:20:38.256768] [264669] [0x79693d5a66c0] [info] Validating backup 1-full
 chunk #6 out of 8
[2025-08-05 16:20:38.278902] [264669] [0x79693d5a66c0] [info] Validating backup 1-full
 chunk #7 out of 8
[2025-08-05 16:20:38.300380] [264669] [0x79693fc1a4c0] [info] Validate time 235ms
[2025-08-05 16:20:38.301628] [264669] [0x79693d5a66c0] [info] Validating backup 1-delta
 chunk #0 out of 8
[2025-08-05 16:20:38.305281] [264669] [0x79693d5a66c0] [info] Validating backup 1-delta
 chunk #1 out of 8
[2025-08-05 16:20:38.368129] [264669] [0x79693d5a66c0] [info] Validating backup 1-delta
 chunk #2 out of 8
[2025-08-05 16:20:38.371719] [264669] [0x79693d5a66c0] [info] Validating backup 1-delta
 chunk #3 out of 8
[2025-08-05 16:20:38.375494] [264669] [0x79693d5a66c0] [info] Validating backup 1-delta
 chunk #4 out of 8
[2025-08-05 16:20:38.379185] [264669] [0x79693d5a66c0] [info] Validating backup 1-delta
 chunk #5 out of 8
[2025-08-05 16:20:38.383215] [264669] [0x79693d5a66c0] [info] Validating backup 1-delta
 chunk #6 out of 8
[2025-08-05 16:20:38.386733] [264669] [0x79693d5a66c0] [info] Validating backup 1-delta
 chunk #7 out of 8
```

```
[2025-08-05 16:20:38.390220] [264669] [0x79693fc1a4c0] [info] Validate time 89ms
[2025-08-05 16:20:38.391428] [264669] [0x79693d5a66c0] [info] Validating backup 2-delta
 chunk #0 out of 8
[2025-08-05 16:20:38.395259] [264669] [0x79693d5a66c0] [info] Validating backup 2-delta
 chunk #1 out of 8
[2025-08-05 16:20:38.399093] [264669] [0x79693d5a66c0] [info] Validating backup 2-delta
 chunk #2 out of 8
[2025-08-05 16:20:38.402872] [264669] [0x79693d5a66c0] [info] Validating backup 2-delta
 chunk #3 out of 8
[2025-08-05 16:20:38.405935] [264669] [0x79693d5a66c0] [info] Validating backup 2-delta
 chunk #4 out of 8
[2025-08-05 16:20:38.468891] [264669] [0x79693d5a66c0] [info] Validating backup 2-delta
 chunk #5 out of 8
[2025-08-05 16:20:38.472336] [264669] [0x79693d5a66c0] [info] Validating backup 2-delta
 chunk #6 out of 8
[2025-08-05 16:20:38.475977] [264669] [0x79693d5a66c0] [info] Validating backup 2-delta
 chunk #7 out of 8
[2025-08-05 16:20:38.479477] [264669] [0x79693fc1a4c0] [info] Validate time 88ms
[2025-08-05 16:20:38.479859] [264669] [0x79693fc1a4c0] [info] INFO: Backup 2-delta is
 valid
[2025-08-05 16:20:38.479992] [264669] [0x79693fc1a4c0] [info] Start restore of backup
 2-delta into /mnt/node
[2025-08-05 16:20:38.481239] [264669] [0x79693fc1a4c0] [info] Backup 1-delta is chosen
 as shiftpoint, its Stop LSN will be used as shift LSN
[2025-08-05 16:20:38.483006] [264669] [0x79693d5a66c0] [info] Restoring 2-delta chunk
 #0 out of 8
[2025-08-05 16:20:38.532219] [264669] [0x79693d5a66c0] [info] Restoring 2-delta chunk
 #1 out of 8
[2025-08-05 16:20:38.569631] [264669] [0x79693d5a66c0] [info] Restoring 2-delta chunk
 #2 out of 8
[2025-08-05 16:20:38.608792] [264669] [0x79693d5a66c0] [info] Restoring 2-delta chunk
 #3 out of 8
[2025-08-05 16:20:38.633202] [264669] [0x79693d5a66c0] [info] Restoring 2-delta chunk
 #4 out of 8
[2025-08-05 16:20:38.733030] [264669] [0x79693d5a66c0] [info] Restoring 2-delta chunk
 #5 out of 8
[2025-08-05 16:20:38.764890] [264669] [0x79693d5a66c0] [info] Restoring 2-delta chunk
 #6 out of 8
[2025-08-05 16:20:38.804717] [264669] [0x79693d5a66c0] [info] Restoring 2-delta chunk
 #7 out of 8
[2025-08-05 16:20:38.839108] [264669] [0x79693fc1a4c0] [info] Restore time 356ms
[2025-08-05 16:20:38.839256] [264669] [0x79693fc1a4c0] [info] Removing redundant files
 in destination directory
[2025-08-05 16:20:38.848171] [264669] [0x79693d5a66c0] [info] Restoring 1-delta chunk
 #0 out of 8
[2025-08-05 16:20:38.860342] [264669] [0x79693d5a66c0] [info] Restoring 1-delta chunk
 #1 out of 8
[2025-08-05 16:20:38.918134] [264669] [0x79693d5a66c0] [info] Restoring 1-delta chunk
 #2 out of 8
[2025-08-05 16:20:38.929649] [264669] [0x79693d5a66c0] [info] Restoring 1-delta chunk
 #3 out of 8
[2025-08-05 16:20:38.942811] [264669] [0x79693d5a66c0] [info] Restoring 1-delta chunk
 #4 out of 8
[2025-08-05 16:20:38.954785] [264669] [0x79693d5a66c0] [info] Restoring 1-delta chunk
 #5 out of 8
[2025-08-05 16:20:38.970253] [264669] [0x79693d5a66c0] [info] Restoring 1-delta chunk
 #6 out of 8
```

```
[2025-08-05 16:20:38.983111] [264669] [0x79693d5a66c0] [info] Restoring 1-delta chunk
 #7 out of 8
[2025-08-05 16:20:38.995516] [264669] [0x79693fc1a4c0] [info] Restore time 148ms
[2025-08-05 16:20:38.997560] [264669] [0x79693d5a66c0] [info] Restoring 1-full chunk #0
 out of 8
[2025-08-05 16:20:39.032484] [264669] [0x79693d5a66c0] [info] Restoring 1-full chunk #1
 out of 8
[2025-08-05 16:20:39.062668] [264669] [0x79693d5a66c0] [info] Restoring 1-full chunk #2
 out of 8
[2025-08-05 16:20:39.134805] [264669] [0x79693d5a66c0] [info] Restoring 1-full chunk #3
 out of 8
[2025-08-05 16:20:39.160183] [264669] [0x79693d5a66c0] [info] Restoring 1-full chunk #4
 out of 8
[2025-08-05 16:20:39.189998] [264669] [0x79693d5a66c0] [info] Restoring 1-full chunk #5
 out of 8
[2025-08-05 16:20:39.219022] [264669] [0x79693d5a66c0] [info] Restoring 1-full chunk #6
 out of 8
[2025-08-05 16:20:39.249562] [264669] [0x79693d5a66c0] [info] Restoring 1-full chunk #7
 out of 8
[2025-08-05 16:20:39.279275] [264669] [0x79693fc1a4c0] [info] Restore time 282ms
[2025-08-05 16:20:39.280742] [264669] [0x79693fc1a4c0] [info] INFO: Restore of backup
 2-delta completed.
```

3.2.3. Partial Restore

You can restore particular databases using partial restore options with the restore command. The sections below describe all supported partial restore methods.

3.2.3.1. Partial Restore by Name

If you have enabled partial restore before taking backups, you can restore specific databases by name using the `--db-include-name` and `--db-exclude-name` options.

To restore only the specified databases, run the restore command with the following options:

```
pg_probackup3 restore -B backup_dir --instance=instance_name --db-include-name=dbname
```

The `--db-include-name` option can be specified multiple times. For example, to restore only the databases `db1` and `db2`, run the following command:

```
pg_probackup3 restore -B backup_dir --instance=instance_name --db-include-name=db1 --
db-include-name=db2
```

To exclude one or more databases from restore, use the `--db-exclude-name` option:

```
pg_probackup3 restore -B backup_dir --instance=instance_name --db-exclude-name=dbname
```

The `--db-exclude-name` option can be specified multiple times. For example, to exclude the databases `db1` and `db2`, run the following command:

```
pg_probackup3 restore -B backup_dir --instance=instance_name --db-exclude-name=db1 --
db-exclude-name=db2
```

> **Note**
>
> After the Postgres Pro cluster is successfully started, drop the excluded databases using the `DROP DATABASE` command.

To decouple a single cluster containing multiple databases into separate clusters with minimal downtime, run partial restore of the cluster as a standby using the `--restore-as-replica` option for specific databases.

> **Note**
>
> The `template0` and `template1` databases are always restored.

> **Warning**
>
> Options `--db-exclude-name` and `--db-include-name` cannot be used together.

3.2.3.2. Partial Backup and Restore by OID

You can backup and restore particular databases *without any special preparations* using the `--db-include-oid` and `--db-exclude-oid`.

> **Note**
>
> Partial backup by OID is currently supported only in the DIRECT mode.

To restore the specified databases only, run the restore command with the following options:

```
pg_probackup3 restore -B backup_dir --instance=instance_name --db-include-oid=dboid
```

> **Note**
>
> After a partial backup, you can perform a non-partial restore, which will result in an instance with only the included databases.

During restore, only WAL records for the included databases are processed, ignoring others, which speeds up the operation.

The `--db-include-oid` option can be specified multiple times. For example, to restore only the `db1` and `db2` databases with OIDs `dboid1` and `dboid2`, respectively, run the following command:

```
pg_probackup3 restore -B backup_dir --instance=instance_name --db-include-oid=dboid1 --db-include-oid=dboid2
```

To exclude one or more databases from restore, use the `--db-exclude-oid` option:

```
pg_probackup3 restore -B backup_dir --instance=instance_name --db-exclude-oid=dboid
```

The `--db-exclude-oid` option can be specified multiple times. For example, to exclude the `db1` and `db2` databases with OIDs `dboid1` and `dboid2`, respectively, from restore, run the following command:

```
pg_probackup3 restore -B backup_dir --instance=instance_name --db-exclude-oid=dboid1 --db-exclude-oid=dboid2
```

> **Note**
>
> After the Postgres Pro cluster is successfully started, drop the excluded databases using the `DROP DATABASE` command.

To decouple a single cluster containing multiple databases into separate clusters with minimal downtime, run partial restore of the cluster as a standby using the `--restore-as-replica` option for specific databases.

> **Note**
>
> The `template0` and `template1` databases are always restored.

> **Warning**
>
> Options `--db-exclude-oid` and `--db-include-oid` cannot be used together.

3.2.4. Performing Point-in-Time Recovery (PITR)

You can restore the cluster to its state at an arbitrary point in time (recovery target) using recovery target options with the restore command.

Before starting PITR, make sure the following conditions are met:

- You have configured continuous WAL archiving with the `wal_level` parameter set to `replica` before taking backups.
- You are taking regular full and incremental backups using pg_probackup3.

You can use both STREAM and ARCHIVE backups for point-in-time recovery as long as the WAL archive contains an unbroken sequence of segments from the backup time to your recovery target time.

Follow the steps below to restore the cluster state at the exact point in time. Use your own values when applicable.

1. Stop the server:

   ```
   pg_ctl stop -D /path/to/database/data
   ```

2. Remove the data directory (`PGDATA`):

   ```
   rm -rf /path/to/database/data
   mv -f /path/to/database/logs/pg_logfile.log /tmp/demo/logs/pg_logfile.log.old
   ```

   This step is optional, as you can use a different directory for restore.

   The log file preservation is recommended for diagnostic purposes.

3. Run the `restore` command with the following options:

   ```
   pg_probackup3 restore -B backup_dir --instance=instance_name --recovery-target-
   time="2024-04-10 18:18:26+03"
   --recovery-target-action=promote
   ```

4. Start the server:

   ```
   pg_ctl start -D /path/to/database/data
   ```

After the recovery is complete, a new instance will be automatically promoted to primary, and a new timeline will be created.

Refer to the section called “Recovery Target Options” for other supported PITR methods.

3.3. Managing the Backup Catalog

With pg_probackup3, you can manage backups from the command line:

- Use the `show` command to view backup information. See the section Viewing Backup Information for more details.
- To view information about the WAL archive, run the `show` command with the `--archive` option. Refer to the section Viewing WAL Archive Information for details.
- Depending on whether you want to merge an incremental backup to its parent full backup or merge a chain of incremental backups, you can use the `merge` command with the `--backup-id`(`-i`) option or `--backup-id` and `--merge-from-id`/`--merge-interval`, respectively. See the section Merging Backups for more information.
- To remove backups that are no longer needed, run the `delete` command. Refer to the section Deleting Backups for more details.

3.3.1. Viewing Backup Information

To view the list of existing backups for every instance, run the command:

```
pg_probackup3 show -B backup_dir
```

pg_probackup3 displays the list of all the available backups. For example:

```
BACKUP INSTANCE 'dev', version 3
============================================================================================================
Instance Version ID       End time                 Mode  WAL Mode TLI Duration Data WAL
 Zalg Zratio Start LSN  Stop LSN   Status
============================================================================================================
dev      17      1-full   2024-12-10 14:51:34+0000 FULL  STREAM   1   1s       38MB -
 none 1.00   0/A000028  0/A000138  OK
dev      17      1-delta  2024-12-10 14:52:02+0000 DELTA STREAM   1            11MB -
 none 1.00   0/D000028  0/D000180  OK
dev      17      2-delta  2024-12-10 14:52:28+0000 DELTA STREAM   1            22MB -
 none 1.00   0/10000028 0/10000138 OK
dev      17      1a-full  2024-12-10 14:54:10+0000 FULL  ARCHIVE  1   1s       75MB -
 none 1.00   0/12000028 0/12000138 OK
dev      17      1a-delta 2024-12-10 14:54:32+0000 DELTA ARCHIVE  1            17MB -
 none 1.00   0/14000028 0/14000138 OK
```

For each backup, the following information is provided:

- `Instance` — the instance name.
- `Version` — Postgres Pro major version.
- `ID` — the backup identifier.
- `End time` — the backup end time.
- `Mode` — the method used to take this backup. Possible values: `FULL`, `DELTA`, `PTRACK`.
- `WAL Mode` — WAL delivery mode. Possible values: `STREAM` and `ARCHIVE`.
- `TLI` — timeline identifiers of the current backup and its parent.
- `Duration` — the time it took to perform the backup.
- `Data` — the size of the data files in this backup. This value does not include the size of WAL files. For STREAM backups, the total size of the backup can be calculated as `Data` + `WAL`.
- `WAL` — the uncompressed size of WAL files that need to be applied during recovery for the backup to reach a consistent state.
- `compress-alg` — compression algorithm used during backup. Possible values: `zlib`, `lz4`, `zstd`, `none`.
- `Zratio` — compression ratio calculated as “uncompressed-bytes” / “data-bytes”.
- `Start LSN` — WAL log sequence number corresponding to the start of the backup process. REDO point for Postgres Pro recovery process to start from.

- `Stop LSN` — WAL log sequence number corresponding to the end of the backup process. Consistency point for Postgres Pro recovery process.
- `Status` — backup status. Possible values:
  - `OK` — the backup is complete and valid.
  - `DONE` — the backup is complete, but was not validated.
  - `RUNNING` — the backup is in progress.
  - `MERGING` — the backup is being merged.
  - `MERGED` — the backup data files were successfully merged, but its metadata is in the process of being updated. Only full backups can have this status.
  - `DELETING` — the backup files are being deleted.
  - `CORRUPT` — some of the backup files are corrupt.
  - `ERROR` — the backup was aborted because of an unexpected error.
  - `ORPHAN` — the backup is invalid because one of its parent backups is corrupt or missing.
  - `HIDDEN_FOR_TEST` — a test script marked the backup as nonexistent. (pg_probackup3 never sets this status by itself.)

You can restore the cluster from the backup only if the backup status is `OK` or `DONE`.

To get more detailed information about the backup, run the `show` command with the backup ID:

```
pg_probackup3 show -B backup_dir --instance=instance_name -i backup_id
```

The sample output is as follows:

```
# Backup 2-delta information.
backup_id=2-delta
parent_backup_id=1-delta
backup_mode=delta
tli=1
start_lsn=268435496
stop_lsn=268435768
# start-time 2024-12-10 14:52:28+0000
start_time=1733842348
# end-time 2024-12-10 14:52:28+0000
end_time=1733842348
recovery-time=0
data-bytes=22986632
uncompressed-bytes=22986632
compress-alg=none
compress-level=1
server-version=170001
min_xid=0
min_multixact=0
backup_source=pro
primary_conninfo=user=garbuz reusepass=1 channel_binding=prefer host=localhost
 port=5432 sslmode=prefer sslcompression=0 sslcertmode=allow sslsni=1
 ssl_min_protocol_version=TLSv1.2 gssencmode=disable krbsrvname=postgres
 gssdelegation=0 target_session_attrs=any target_server_type=any hostorder=sequential
 load_balance_hosts=disable
stream=true
program-version=3.0.0
block-size=8192
xlog-block-size=8192
status = OK
```

Detailed output has additional attributes:

- `compress-alg` — compression algorithm used during backup. Possible values: `zlib`, `lz4`, `zstd`, `none`.

- `compress-level` — compression level used during backup.
- `block-size` — the *block_size* setting of Postgres Pro cluster at the backup start.
- `checksum-version` — are *data block checksums* enabled in the backed up Postgres Pro cluster. Possible values: `1`, `0`.
- `program-version` — full version of pg_probackup3 binary used to create the backup.
- `start-time` — the backup start time.
- `end-time` — the backup end time.
- `end-validation-time` — the backup validation end time.
- `expire-time` — the point in time when a pinned backup can be removed in accordance with retention policy. This attribute is only available for pinned backups.
- `uncompressed-bytes` — the size of data files before adding page headers and applying compression. You can evaluate the effectiveness of compression by comparing `uncompressed-bytes` to `data-bytes` if compression if used.
- `data-bytes` — the size of Postgres Pro cluster data files at the time of backup. You can evaluate the effectiveness of an incremental backup by comparing `data-bytes` to `uncompressed-bytes`.
- `recovery-xid` — transaction ID at the backup end time.
- `parent-backup-id` — ID of the parent backup. Available only for incremental backups.
- `primary_conninfo` — libpq connection parameters used to connect to the Postgres Pro cluster to take this backup. The password is not included.
- `note` — text note attached to backup.
- `content-crc` — CRC32 checksum of `backup_content.control` file. It is used to detect corruption of backup metainformation.

You can use the `--format=tree` option to see the list of backups as a tree:

```
pg_probackup3 show -B backup_dir --format=tree
```

The sample output will look as follows:

```
BACKUP INSTANCE 'dev', version 3

├── 1-full

│   └── 1-delta

│       └── 2-delta

└── 1a-full

    └── 1a-delta
```

You can also get the detailed information about the backup in the JSON format:

```
pg_probackup3 show -B backup_dir --instance=instance_name --format=json -i backup_id
```

The sample output is as follows:

```
[
    {
        "instance": "dev",
        "backups": [
            {
                "id": "2-delta",
                "parent-backup-id": "1-delta",
                "status": "OK",
                "start-time": "2024-12-10 14:52:28+0000",
                "end-time": "2024-12-10 14:52:28+0000",
                "backup-mode": "DELTA",
                "wal": "STREAM",
```

```
                "block-size": 8192,
                "xlog-block-size": 8192,
                "program-version": "3.0.0",
                "server-version": 17,
                "current-tli": 1,
                "start-lsn": "0/10000028",
                "stop-lsn": "0/10000138",
                "data-bytes": 22986632,
                "uncompressed-bytes": 22986632,
                "wal-bytes": 0,
                "compress-alg": "none",
                "compress-level": 1,
                "min-xid": 0,
                "min-multixact": 0,
                "backup-source": "pro"
            }
        ]
    }
]
```


3.3.2. Viewing WAL Archive Information

To view the information about WAL archive for every instance, run the command:

```
pg_probackup3 show -B backup_dir [--instance=instance_name] --archive
```

pg_probackup3 displays the list of all the available WAL files grouped by timelines. For example:

```
BACKUP INSTANCE 'dev', version 3
============================================================================================================
TLI Parent TLI Switchpoint Min Segno                Max Segno                N segments
 Size Zratio N backups Status
============================================================================================================
1              0/0         000000010000000000000001 000000010000000000000006 6
 96MB 1.17   1          OK
```

For each timeline, the following information is provided:

- `TLI` — timeline identifier.
- `Parent TLI` — identifier of the timeline from which this timeline branched off.
- `Switchpoint` — LSN of the moment when the timeline branched off from its parent timeline.
- `Min Segno` — the first WAL segment belonging to the timeline.
- `Max Segno` — the last WAL segment belonging to the timeline.
- `N segments` — number of WAL segments belonging to the timeline.
- `Size` — the size that files take on disk.
- `Zalg` — compression algorithm used during backup. Possible values: `zlib`, `lz4`, `zstd`, `none`.
- `Zratio` — compression ratio calculated as `N segments` * *`wal_segment_size`* * *`wal_block_size`* / `Size`.
- `N backups` — number of backups belonging to the timeline. To get the details about backups, use the JSON format.
- `Status` — status of the WAL archive for this timeline. Possible values:
  - `OK` — all WAL segments between `Min Segno` and `Max Segno` are present.
  - `DEGRADED` — some WAL segments between `Min Segno` and `Max Segno` are missing. To find out which files are lost, view this report in the JSON format. This status may appear if several WAL files (in the middle of the sequence) were deleted by the delete command with the `--delete-wal` option according to the retention policy. This status does not affect the restore correctness, but it can be impossible to perform PITR of the cluster to some recovery targets.

To get more detailed information about the WAL archive in the JSON format, run the command:

```
pg_probackup3 show -B backup_dir [--instance=instance_name] --archive --format=json
```

The sample output is as follows:


```
[
    {
        "instance": "dev",
        "version": "3",
        "timelines": [
            {
                "tli": 1,
                "parent-tli": 0,
                "switchpoint": "0/0",
                "min-segno": "000000010000000000000001",
                "max-segno": "000000010000000000000006",
                "n-segments": 6,
                "size": 100663615,
                "zratio": 1.17,
                "status": "OK",
                "backups": [
                    {
                        "id": "1-full",
                        "status": "OK",
                        "start-time": "2025-02-11 14:22:16+0000",
                        "end-time": "2025-02-11 14:22:16+0000",
                        "backup-mode": "FULL",
                        "wal": "STREAM",
                        "block-size": 8192,
                        "xlog-block-size": 8192,
                        "program-version": "3.0.0",
                        "server-version": 17,
                        "current-tli": 1,
                        "start-lsn": "0/5000028",
                        "stop-lsn": "0/5000128",
                        "data-bytes": 60748163,
                        "uncompressed-bytes": 60748163,
                        "wal-bytes": 0,
                        "compress-alg": "none",
                        "compress-level": 1,
                        "min-xid": 0,
                        "min-multixact": 0,
                        "backup-source": "pro"
                    }
                ]
            }
        ]
    }
]
```


Most fields are consistent with the plain format, with some exceptions:

- The size is in bytes.
- The `closest-backup-id` attribute contains the ID of the most recent valid backup that belongs to one of the previous timelines. You can use this backup to perform point-in-time recovery to this timeline. If such a backup does not exist, this string is empty.
- The `lost-segments` array provides with information about intervals of missing segments in `DEGRADED` timelines. In `OK` timelines, the `lost-segments` array is empty.
- The `backups` array lists all backups belonging to the timeline. If the timeline has no backups, this array is empty.

3.3.3. Merging Backups

As you take more and more incremental backups, the total size of the backup catalog can substantially grow. To save disk space, you can merge incremental backups to their parent full backups or merge chains of incremental backups.

During the merge, a brand-new backup is created, into which all the backups to be merged are added. All redundant backups are deleted *only after* the merge is successful. While this process requires additional disk space, it helps prevent data loss in case of any errors or system failures.

> **Note**
>
> If several child backups relate to the same parent, such backups are not deleted after merge, and the disk space is not freed.

To merge an incremental backup to its parent full backup, run the `merge` command, specifying the backup ID of the most recent incremental backup you would like to merge:

```
pg_probackup3 merge -B backup_dir --instance=instance_name -i backup_id
```

This command merges backups that belong to a common incremental backup chain. If you specify a full backup, it will be merged with its first incremental backup. If you specify an incremental backup, it will be merged to its parent full backup, together with all incremental backups between them. Once the merge is complete, the full backup takes in all the merged data, and the incremental backups are removed as redundant. Thus, the merge operation is virtually equivalent to retaking a full backup and removing all the outdated backups, but it allows you to save much time, especially for large data volumes, as well as I/O and network traffic if you are using pg_probackup3 in the remote mode.

To merge a chain of incremental backups, excluding a full backup, specify the IDs of the first and the last incremental backup in the chain:

```
pg_probackup3 merge -B backup_dir --instance=instance_name --merge-from-id=merge_from -
i backup_id
```

Or specify the first backup ID followed by the time interval (in hours) to merge all the backups created during this time:

```
pg_probackup3 merge -B backup_dir --instance=instance_name -i backup_id --merge-
interval=merge_interval
```

Before the merge, pg_probackup3 validates all the affected backups to ensure that they are valid. You can check the current backup status by running the show command with the backup ID:

```
pg_probackup3 show -B backup_dir --instance=instance_name -i backup_id
```

If the merge is still in progress, the backup status is displayed as `MERGING`. For full backups, it can also be shown as `MERGED` while the metadata is being updated at the final stage of the merge. The merge is idempotent, so you can restart the merge if it was interrupted.

> **Warning**
>
> Avoid force-terminating the merge operation, as it may cause subsequent `merge` commands to fail and disrupt backup validation.

3.3.4. Deleting Backups

To delete a backup that is no longer required, run the following command:

```
pg_probackup3 delete -B backup_dir --instance=instance_name -i backup_id
```

This command will delete the backup with the specified *backup_id*, together with all the incremental backups that descend from *backup_id*, if any. This way you can delete some recent incremental backups, retaining the underlying full backup and some of the incremental backups that follow it.

Before deleting backups, you can run the `delete` command with the `--dry-run` flag, which displays the status of all the available backups according to the current retention policy, without performing any irreversible actions.

To delete all backups with a specific status, use the `--status` option:

```
pg_probackup3 delete -B backup_dir --instance=instance_name --status=ERROR
```

Deleting backups by status ignores established retention policies.

3.4. Using pg_probackup3 in the Remote Mode

pg_probackup3 supports operations in the remote mode. The backup data can be obtained from a remote Postgres Pro server using the built-in replication protocol. Backups can be saved to an S3 storage or to an SSH server via the SFTP protocol.

Depending on the type of operation you can choose the following scenarios:

- To connect to the Postgres Pro server to create a backup, use the standard connection options: `--pghost`, `--pgport`, `--pgdatabase`, and `--pguser`.
- To save, restore, and validate backups in an S3 storage, use the S3 options. Refer to Section 2.12 for more details.
- To manage backups on an SSH server via the SFTP protocol, use the SSH options. Refer to Section 2.11 for more details.

This way, both the backup catalog and the Postgres Pro instance to back up can be located on remote servers.

The typical workflow is as follows:

- On your backup host, configure pg_probackup3 as explained in the section Installation and Setup. For the init, add-instance, backup, and set-config commands, make sure to specify the connection options to the server with the Postgres Pro instance, as well as the the options to access the storage where the backups will be saved.
- If you would like to rely on ARCHIVE WAL delivery mode, configure continuous WAL archiving from the database host as explained in the section Setting up continuous WAL archiving. For the archive-push command, specify the SSH options that point to the backup host with the backup catalog.

> **Note**
>
> To use the continuous WAL archiving mode, the pg_probackup3 executable files must be located on the Postgres Pro server. The Postgres Pro server calls pg_probackup3 with the `archive-push` and `archive-get` commands in accordance with the *archive_command* and *restore_command* parameters in its configuration.

For example, to create a **local** archive full backup of a Postgres Pro cluster located on a remote system with the host address `192.168.0.2` on behalf of the `postgres` user through the port `2302`, run:

```
pg_probackup3 backup -B backup_dir --instance=instance_name -b FULL --pguser=postgres
 --pghost=192.168.0.2 --pgport=2302
```

To create a backup on an SSH server with the address `10.0.3.77` for the `ubuntu` user, run:

```
pg_probackup3 backup -B backup_dir --instance=instance_name -b FULL --pguser=postgres
 --pghost=192.168.0.2 --pgport=2302 --remote-host=10.0.3.77 --remote-user=ubuntu
```

The script for creating a backup in an S3 storage is as follows:

```
export PG_PROBACKUP_S3_PORT=9000
export PG_PROBACKUP_S3_ACCESS_KEY=admin
export PG_PROBACKUP_S3_SECRET_ACCESS_KEY=password
export PG_PROBACKUP_S3_REGION=us-west-2
export PG_PROBACKUP_S3_HOST=10.0.3.77
export PG_PROBACKUP_S3_BUCKET_NAME=test
pg_probackup3 backup -B backup_dir --instance=instance_name -b FULL --pguser=postgres
 --pghost=192.168.0.2 --pgport=2302 --s3
```

To set up continuous WAL archiving to an SSH server with the address `10.0.3.77`, run the following command:

```
pg_probackup3 archive-push -B backup_dir --instance=instance_name --wal-file-name=%f --
compress-algorithm=zstd --remote-host=10.0.3.77 --remote-user=ubuntu
```

To set up continuous WAL archiving to an S3 storage, run the following command:

```
pg_probackup3 archive-push -B backup_dir --instance=instance_name --compress-
algorithm=zstd --wal-file-name=%f --s3 --config-file=s3.config
```

The connection options must be saved in the `s3.config` file in the Postgres Pro data directory.

3.5. Remote Restore

Remote restore in pg_probackup3 is a functionality that allows restoring backups directly to a remote host without the need to manually copy archive files. This approach significantly reduces recovery time and minimizes manual operations during disaster recovery, migration, or automated deployment. It is particularly useful in infrastructures with centralized backup storage where recovery is performed onto isolated remote servers.

The remote restore functionality consists of the following main components:

- The send-backup command in the pg_probackup3 utility to send data to a remote server through a specified port using multithreading.
- The pgpro_backupstream utility, manually started on the remote server, to receive, extract, and restore the data.

The following conditions must be met to perform remote restore:

- On the local system:
  - The pg_probackup3 utility and the `libpgprobackup` library must be installed.
  - The backup catalog must be accessible.
- On the remote system:
  - The pgpro_backupstream utility must be installed.

    Note

    pgpro_backupstream requires manual startup.

  - The `PGDATA` directory must be empty and writable.

    Note

    The incremental restore mode is not currently supported.

- The specified port must be open and available for incoming connections.

To restore an instance to a remote host, perform the following steps:

1. On the local system's side, run the send-backup command via the pg_probackup3 utility to send the backup data to the remote host over the specified port:

   ```
   pg_probackup3 send-backup -B backup_dir --instance=instance_name -i backup_id -
   p port -h host [--no-merge]
   ```

   Using the `--no-merge` flag will prevent the backup chain from being merged before the data transfer. Otherwise, the backup chain will be merged into a temporary file, which will be deleted after the transfer is complete.

2. On the remote system's side, run the `restore` command via the pgpro_backupstream utility to receive the backup data and perform restore:

   ```
   pgpro_backupstream restore -D path_to_restore [-p port]
   ```

   If the port is not specified, STDIN is used.

Important

Run the pgpro_backupstream utility on the remote system **before** initiating the data transfer with the send-backup command.

3.6. Running pg_probackup3 on Parallel Threads

backup, restore, merge, delete, catchup, and validate processes can be executed on several parallel threads. This can significantly speed up pg_probackup3 operation given enough resources (CPU cores, disk, and network bandwidth).

Parallel execution is controlled by the `-j`/`--threads`, `--num-write-threads`, and `--num-validate-threads` command-line options. These options must be non-negative integers.

If these options are not specified or set to zero, pg_probackup3 defaults to the number of CPU cores. If the core count cannot be determined, a single thread will be used.

When specified, `--num-write-threads` and `--num-validate-threads` override `-j`.

If the requested threads exceed the system limit (e.g., from `/proc/sys/kernel/threads-max`), a warning will be displayed, and the system limit value will be used instead. If no limit is found, the value specified by the user will be applied.

In the PRO mode, the number of read threads must be less than the value of the `max_wal_senders` server parameter.

For example, to create a backup using four parallel threads, run the following command:

```
pg_probackup3 backup -B backup_dir --instance=instance_name -b FULL -j 4
```

Note

Parallel restore applies only to copying data from the backup catalog to the data directory of the cluster. When Postgres Pro server is started, WAL records need to be replayed, and this cannot be done in parallel.

3.7. Mounting a Backup Directory with FUSE

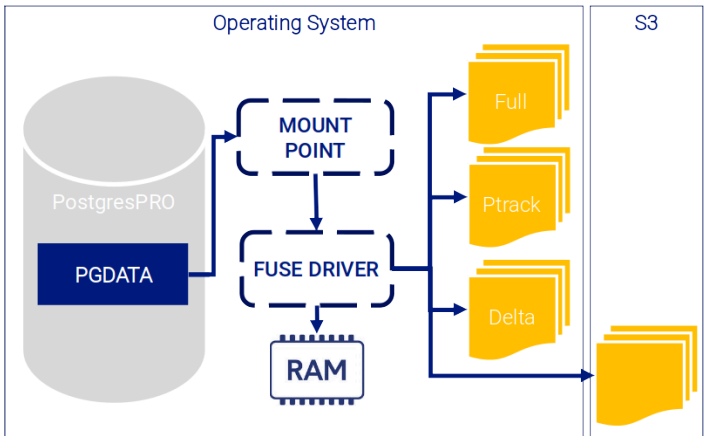
pg_probackup3 allows running a database instance directly from a backup, inspecting and restoring specific data without requiring a full restore, using the fuse command.

This command implements the FUSE (Filesystem in User Space) mechanism, mounting a virtual representation of the backup directory. Postgres Pro interacts with this mounted directory as if it were an actual `PGDATA` directory, while proxying all file system requests to the backup files. Since changes are written to the cache rather than the backup, the original backup remains unchanged, and all operations are *read-only*.

> **Note**
>
> `fuse` operations are only available in Postgres Pro Enterprise versions and editions.

> **Warning**
>
> The FUSE mechanism does not provide all the capabilities required for production environment. It is thus recommended to use `fuse` for one-time operations only. The performance of `fuse` operations may vary significantly depending on your storage type (S3, NFS, SSD).

Figure 3.1. The pg_probackup3 FUSE Mechanism



The key use cases for the fuse command are as follows:

- Restore deleted data from a particular date (for example, using pg_dump).
- Investigate data from a certain point in time.
- Provide a read-only production-like environment when a full restore would be time-consuming.
- Roll back to a specific moment in time to test and debug application failures.
- Run reports on a backup without the overhead of a full restore, as an alternative to replication.
- Support developer databases on FUSE without the need to perform a full multi-gigabyte restore.

> **Note**
>
> On ALT Linux, the user running pg_probackup3 must be a member of the `fuse` group. See the *ALT Linux documentation* for details.

To use the mounted backup as `PGDATA`, set *`mnt_path`* as the path for the `-D` parameter when starting Postgres Pro with `pg_ctl start`.

To ensure sufficient disk space for FUSE operations, use the `--cache-dir` option to specify a custom directory for cache storage.

Mounting a backup chain requires pre-generated file maps. To enable file map generation, use one of the following methods:

- Use the `--with-file-map` option with the backup or merge command.
- Run the file-map command for an existing backup chain. Note that this will replace any previously generated file maps.

For automation purposes, you can run the FUSE filesystem in the background by using the `--detach` option.

To unmount the FUSE filesystem, run the `fuse` command with the `--unmount` option:

```
pg_probackup3 fuse --unmount --mnt-path=mnt_path
```

For details on the `fuse` command and its parameters, refer to the section called “Commands”.

3.8. Checking Data Integrity

3.8.1. Page Validation

If *data checksums* are enabled in the database cluster, pg_probackup3 uses this information to check correctness of data files during backup. While reading each page, pg_probackup3 checks whether the calculated checksum coincides with the checksum stored in the page header. This guarantees that the Postgres Pro instance and the backup itself have no corrupt pages. Note that pg_probackup3 reads database files directly from the filesystem, so under heavy write load during backup it can show false-positive checksum mismatches because of partial writes. If a page checksum mismatch occurs, the page is re-read and checksum comparison is repeated.

A page is considered corrupt if checksum comparison has failed more than 300 times. In this case, the backup is aborted.

Even if data checksums are not enabled, pg_probackup3 always performs sanity checks for page headers.

3.9. Configuring Retention Policy

With pg_probackup3, you can configure retention policy to remove redundant backups, clean up unneeded WAL files, as well as pin specific backups to ensure they are kept for the specified time, as explained in the sections below. All these actions can be combined together in any way.

3.9.1. Removing Redundant Backups

By default, all backup copies created with pg_probackup3 are stored in the specified backup catalog. To save disk space, you can configure retention policy to remove redundant backup copies.

To configure retention policy, set one or more of the following variables in the `pg_probackup3.conf` file via set-config:

`--retention-redundancy=`*`redundancy`*

Specifies **the number of full backup copies** to keep in the backup catalog.

`--retention-window=`*`window`*

Defines the earliest point in time for which pg_probackup3 can complete the recovery. This option is set in **the number of days** from the current moment. For example, if `retention-window=6`, pg_probackup3 must keep at least one backup copy that is older than six days, with all the corresponding WAL files, and all the backups that follow.

If both `--retention-redundancy` and `--retention-window` options are set, both these conditions have to be taken into account when purging the backup catalog. For example, if you set `--retention-redundancy=2` and `--retention-window=6`, pg_probackup3 has to keep two full backup copies, as well as all the backups required to ensure recoverability for the last six days:

```
pg_probackup3 set-config -B backup_dir --instance=instance_name --retention-
redundancy=2 --retention-window=6
```

It is recommended to always keep at least two last parent full backups to avoid errors when creating incremental backups.

To clean up the backup catalog in accordance with retention policy, you have to run the retention command with retention flags, as shown below.

For example, to remove all backup copies that no longer satisfy the defined retention policy, run the following command with the `--delete-expired` flag:

```
pg_probackup3 retention -B backup_dir --instance=instance_name --delete-expired
```

If you would like to also remove the WAL files that are no longer required for any of the backups, you should also specify the `--delete-wal` flag:

```
pg_probackup3 retention -B backup_dir --instance=instance_name --delete-expired --
delete-wal
```

You can also set or override the current retention policy by specifying `--retention-redundancy` and `--retention-window` options directly when running the `retention` command:

```
pg_probackup3 retention -B backup_dir --instance=instance_name --delete-expired --
retention-window=6 --retention-redundancy=2
```

Since incremental backups require that their parent full backup and all the preceding incremental backups are available, if any of such backups expire, they still cannot be removed while at least one incremental backup in this chain satisfies the retention policy. To avoid keeping expired backups that are still required to restore an active incremental one, you can merge them with this backup using the `--merge-expired` flag when running the retention command.

Suppose you have backed up the *node* instance in the *backup_dir* directory, with the `--retention-window` option set to `6` and `--retention-redundancy` option set to `2`, and you have the following backups available on February 11, 2025:

```
BACKUP INSTANCE 'dev', version 3
============================================================================================
Instance Version ID        End time                 Mode  WAL Mode TLI Duration Data
 WAL Zalg Zratio Start LSN  Stop LSN   Status
============================================================================================
dev      17      full-1    2024-10-18 21:02:28+0000 FULL  ARCHIVE  1            87MB  -
   none 1.00   0/10000028 0/10000128 OK
dev      17      delta-1-1 2024-11-11 00:36:01+0000 DELTA ARCHIVE  1            23MB  -
   none 1.00   0/12000028 0/12000128 OK
dev      17      delta-1-2 2024-11-15 15:43:01+0000 DELTA ARCHIVE  1            22MB  -
   none 1.00   0/14000028 0/14000128 OK
dev      17      full-2    2024-11-22 14:24:04+0000 FULL  ARCHIVE  1            98MB  -
   none 1.00   0/17000028 0/17000128 OK
dev      17      delta-2-1 2024-11-23 18:10:55+0000 DELTA ARCHIVE  1            23MB  -
   none 1.00   0/19000028 0/19000128 OK
 ------------------------------------------------------retention
 window------------------------------------------------------------
dev      17      delta-2-2 2025-02-06 23:44:33+0000 DELTA ARCHIVE  1            33MB  -
   none 1.00   0/1C000028 0/1C000128 OK
dev      17      full-3    2025-02-08 03:31:33+0000 FULL  ARCHIVE  1            120MB -
   none 1.00   0/1F000028 0/1F000128 OK
```

```
dev      17      delta-3-1 2025-02-09 07:18:31+0000 DELTA ARCHIVE  1            23MB  -
   none 1.00   0/21000028 0/21000128 OK
dev      17      delta-3-2 2025-02-10 11:05:17+0000 DELTA ARCHIVE  1            23MB  -
   none 1.00   0/23000028 0/23000128 OK
dev      17      full-4    2025-02-11 15:00:38+0000 FULL  ARCHIVE  1   1s       123MB -
   none 1.00   0/25000028 0/25000128 OK
```

If you run the retention command with the `--delete-expired` flag, the backups with IDs `full-1`, `delta-1-1`, and `delta-1-2` will be removed as they are expired both according to the retention window and due to redundancy (the required set of full backups has already been retained). `delta-1-1` and `delta-1-2` will also be removed since the base full backup is expired.

Running the retention command with the `--merge-expired` flag will merge backups `full-2` and `delta-2-1` with `delta-2-2`. The merge will occur with `delta-2-2` as it is the first non-expired delta backup, which can be merged with expired delta backups `delta-2-1` and expired full backup `full-2`. The new full backup ID will take the value of the current timestamp.

```
pg_probackup3 retention -B backup_dir --instance=node --delete-expired --merge-expired
pg_probackup3 show -B backup_dir
```

```
BACKUP INSTANCE 'dev', version 3
============================================================================================
Instance Version ID                       End time                 Mode  WAL Mode TLI
 Duration Data  WAL Zalg Zratio Start LSN  Stop LSN   Status
============================================================================================
dev      17      2025-02-11-11-14-18-254 2025-02-11 11:14:18+0000 FULL  ARCHIVE  1
       108MB  -  none 1.00   0/17000028 0/19000128 OK
dev      17      full-3                  2025-02-08 03:31:33+0000 FULL  ARCHIVE  1
       120MB -   none 1.00   0/1F000028 0/1F000128 OK
dev      17      delta-3-1               2025-02-09 07:18:31+0000 DELTA ARCHIVE  1
       23MB  -   none 1.00   0/21000028 0/21000128 OK
dev      17      delta-3-2               2025-02-10 11:05:17+0000 DELTA ARCHIVE  1
       23MB  -   none 1.00   0/23000028 0/23000128 OK
dev      17      full-4                  2025-02-11 15:00:38+0000 FULL  ARCHIVE  1   1s
       123MB -   none 1.00   0/25000028 0/25000128 OK
```

The `Duration` field for the merged backup displays the time required for the merge.

3.9.2. Pinning Backups

If you need to keep certain backups longer than the established retention policy allows, you can pin them for arbitrary time. For example:

```
pg_probackup3 set-backup -B backup_dir --instance=instance_name -i backup_id --ttl=30d
```

This command sets the expiration time of the specified backup to 30 days starting from the time indicated in its `recovery-time` attribute.

You can also explicitly set the expiration time for a backup using the `--expire-time` option. For example:

```
pg_probackup3 set-backup -B backup_dir --instance=instance_name -i backup_id --expire-
time="2027-04-09 18:21:32+00"
```

Alternatively, you can use the `--ttl` and `--expire-time` options with the backup command to pin the newly created backup:

```
pg_probackup3 backup -B backup_dir --instance=instance_name -b FULL --ttl=30d
pg_probackup3 backup -B backup_dir --instance=instance_name -b FULL --expire-
time="2027-04-09 18:21:32+00"
```

To check if the backup is pinned, run the show command:

```
pg_probackup3 show -B backup_dir --instance=instance_name -i backup_id
```

If the backup is pinned, it has the `expire-time` attribute that displays its expiration time:

```
...
recovery-time = '2024-04-09 18:21:32+00'
expire-time = '2027-04-09 18:21:32+00'
data-bytes = 22288792
...
```

You can unpin the backup by setting the `--ttl` option to zero:

```
pg_probackup3 set-backup -B backup_dir --instance=instance_name -i backup_id --ttl=0
```

> **Note**
>
> A pinned incremental backup implicitly pins all its parent backups. If you unpin such a backup later, its implicitly pinned parents will also be automatically unpinned.

3.9.3. Configuring WAL Archive Retention Policy

When continuous WAL archiving is enabled, archived WAL segments can take a lot of disk space. Even if you delete old backup copies from time to time, the `--delete-wal` flag can purge only those WAL segments that do not apply to any of the remaining backups in the backup catalog. However, if point-in-time recovery is critical only for the most recent backups, you can configure WAL archive retention policy to keep WAL archive of limited depth and win back some more disk space.

Suppose you have backed up the `node` instance in the *backup_dir* directory and configured continuous WAL archiving:

```
pg_probackup3 show -B backup_dir --instance=node
```

```
BACKUP INSTANCE 'dev', version 3
============================================================================================
Instance Version ID                       End time                 Mode  WAL Mode TLI
 Duration Data  WAL Zalg Zratio Start LSN  Stop LSN   Status
============================================================================================
dev      17      2025-02-11-15-13-36-756 2025-02-11 15:13:37+0000 FULL  ARCHIVE  1   1s
       38MB  -   none 1.00   0/17000028 0/19000128 OK
dev      17      2025-02-11-14-51-12-937 2025-02-06 23:44:33+0000 DELTA ARCHIVE  1
       33MB  -   none 1.00   0/1C000028 0/1C000128 OK
dev      17      2025-02-11-14-51-33-367 2025-02-08 03:31:33+0000 FULL  ARCHIVE  1
       120MB -   none 1.00   0/1F000028 0/1F000128 OK
dev      17      2025-02-11-14-51-51-220 2025-02-09 07:18:31+0000 DELTA ARCHIVE  1
       23MB  -   none 1.00   0/21000028 0/21000128 OK
dev      17      2025-02-11-14-51-57-473 2025-02-10 11:05:17+0000 DELTA ARCHIVE  1
       23MB  -   none 1.00   0/23000028 0/23000128 OK
dev      17      2025-02-11-15-00-37-815 2025-02-11 15:00:38+0000 FULL  ARCHIVE  1   1s
       123MB -   none 1.00   0/25000028 0/25000128 OK
```

You can check the state of the WAL archive by running the show command with the `--archive` flag:

```
pg_probackup3 show -B backup_dir --instance=node --archive
```

```
BACKUP INSTANCE 'dev', version 3
============================================================================================
TLI Parent TLI Switchpoint Min Segno                Max Segno                N segments
 Size  Zratio N backups Status
============================================================================================
1              0/0         000000010000000000000001 000000010000000000000025 37
 592MB 1.41   6         OK
```

To purge all unused WAL files (that do not apply to any of the remaining backups in the backup catalog) run the following command:

```
pg_probackup3 retention -B backup_dir --instance=node --delete-wal
```

```
[2025-02-11 15:23:30.422696] [14218] [128670453549440] [info] command: ./pg_probackup3
 retention -B /work/backup --instance dev --delete-wal
[2025-02-11 15:23:30.422738] [14218] [128670453549440] [info] execute command:
 'retention', instance 'dev'
[2025-02-11 15:23:30.426167] [14218] [128670453549440] [info] WAL file
 000000010000000000000001 removed
[2025-02-11 15:23:30.428095] [14218] [128670453549440] [info] WAL file
 000000010000000000000002 removed
[2025-02-11 15:23:30.429776] [14218] [128670453549440] [info] WAL file
 000000010000000000000003 removed
[2025-02-11 15:23:30.431838] [14218] [128670453549440] [info] WAL file
 000000010000000000000004 removed
[2025-02-11 15:23:30.434124] [14218] [128670453549440] [info] WAL file
 000000010000000000000005 removed
[2025-02-11 15:23:30.434196] [14218] [128670453549440] [info] WAL file
 000000010000000000000005.00000028.backup removed
[2025-02-11 15:23:30.435852] [14218] [128670453549440] [info] WAL file
 000000010000000000000006 removed
[2025-02-11 15:23:30.437579] [14218] [128670453549440] [info] WAL file
 000000010000000000000007 removed
[2025-02-11 15:23:30.441360] [14218] [128670453549440] [info] WAL file
 000000010000000000000008 removed
[2025-02-11 15:23:30.441815] [14218] [128670453549440] [info] WAL file
 000000010000000000000008.00000028.backup removed
[2025-02-11 15:23:30.444488] [14218] [128670453549440] [info] WAL file
 000000010000000000000009 removed
[2025-02-11 15:23:30.446902] [14218] [128670453549440] [info] WAL file
 00000001000000000000000A removed
[2025-02-11 15:23:30.446961] [14218] [128670453549440] [info] WAL file
 00000001000000000000000A.00000028.backup removed
[2025-02-11 15:23:30.448960] [14218] [128670453549440] [info] WAL file
 00000001000000000000000B removed
[2025-02-11 15:23:30.450991] [14218] [128670453549440] [info] WAL file
 00000001000000000000000C removed
[2025-02-11 15:23:30.451069] [14218] [128670453549440] [info] WAL file
 00000001000000000000000C.00000028.backup removed
[2025-02-11 15:23:30.453236] [14218] [128670453549440] [info] WAL file
 00000001000000000000000D removed
[2025-02-11 15:23:30.455291] [14218] [128670453549440] [info] WAL file
 00000001000000000000000E removed
[2025-02-11 15:23:30.455462] [14218] [128670453549440] [info] WAL file
 00000001000000000000000E.00000028.backup removed
[2025-02-11 15:23:30.458088] [14218] [128670453549440] [info] WAL file
 00000001000000000000000F removed
[2025-02-11 15:23:30.459755] [14218] [128670453549440] [info] WAL file
 000000010000000000000010 removed
[2025-02-11 15:23:30.459794] [14218] [128670453549440] [info] WAL file
 000000010000000000000010.00000028.backup removed
[2025-02-11 15:23:30.461135] [14218] [128670453549440] [info] WAL file
 000000010000000000000011 removed
[2025-02-11 15:23:30.462603] [14218] [128670453549440] [info] WAL file
 000000010000000000000012 removed
[2025-02-11 15:23:30.462637] [14218] [128670453549440] [info] WAL file
 000000010000000000000012.00000028.backup removed
```

```
[2025-02-11 15:23:30.464003] [14218] [128670453549440] [info] WAL file
 000000010000000000000013 removed
[2025-02-11 15:23:30.465522] [14218] [128670453549440] [info] WAL file
 000000010000000000000014 removed
[2025-02-11 15:23:30.465555] [14218] [128670453549440] [info] WAL file
 000000010000000000000014.00000028.backup removed
[2025-02-11 15:23:30.466910] [14218] [128670453549440] [info] WAL file
 000000010000000000000015 removed
[2025-02-11 15:23:30.468572] [14218] [128670453549440] [info] WAL file
 000000010000000000000016 removed
[2025-02-11 15:23:30.468600] [14218] [128670453549440] [info] 30 WAL files removed.
```

You can check the state of the WAL archive by running the show command with the `--archive` flag:

```
pg_probackup3 show -B backup_dir --instance=node --archive
```

```
BACKUP INSTANCE 'dev', version 3
============================================================================================
TLI Parent TLI Switchpoint Min Segno                Max Segno                N segments
 Size  Zratio N backups Status
============================================================================================
1              0/0         000000010000000000000017 000000010000000000000025 15
 240MB 1.47   6         OK
```

3.10. Cloning and Synchronizing a Postgres Pro Instance

pg_probackup3 can create a copy of a Postgres Pro instance directly, without using the backup catalog. To do this, you can run the catchup command. It can be useful in the following cases:

- To add a new standby server.

  If the data directory of the destination instance is empty, the `catchup` command works similarly to the `backup` command in the PRO mode, but it can be faster if run in parallel mode.

  > **Note**
  >
  > The `catchup` operations are currently only supported in the PRO mode and do not require direct access to `PGDATA`.

- To have a fallen-behind standby server “catch up” with the primary.

  Under write-intensive load, replicas may fail to replay WAL fast enough to keep up with the primary and hence may lag behind. A usual solution to create a new replica and switch to it requires a lot of extra space and data transfer. The `catchup` command allows you to update an existing replica much faster by fetching differences from the primary.

`catchup` specifications and limitations:

- The backup catalog is not required.
- Copying external directories is not supported.
- The remote mode is not supported.
- DDL commands *`CREATE TABLESPACE`*/*`DROP TABLESPACE`* cannot be run simultaneously with `catchup`.
- `catchup` takes configuration files, such as `postgresql.conf`, `postgresql.auto.conf`, or `pg_hba.conf`, from the source server and overwrites them on the target server. The `--exclude-path` option allows you to keep the configuration files intact.

To prepare for cloning/synchronizing a Postgres Pro instance, set up the source server as follows:

- Configure the database cluster for the instance to copy.
- To use the PTRACK catchup mode, set up PTRACK backups.

Before cloning/synchronizing a Postgres Pro instance, ensure that the source server is running and accepting connections. To clone/synchronize a Postgres Pro instance, on the server with the destination instance, run the catchup command as follows:

```
pg_probackup3 catchup -b catchup_mode --destination-pgdata=path_to_local_dir --stream
 [connection_options]
```

Where *catchup_mode* can take one of the following values:

- `FULL` — creates a full copy of the Postgres Pro instance. The data directory of the destination instance must be empty for this mode.
- `DELTA` — reads all data files in the data directory and creates an incremental copy for pages that have changed since the destination instance was shut down.
- `PTRACK` — tracks page changes on the fly, only reads and copies pages that have changed since the point of divergence of the source and destination instances.

By specifying the `--stream` option, you can set the STREAM WAL delivery mode of copying, which will include all the necessary WAL files by streaming them from the server via replication protocol.

You can use connection_options to specify the connection to the source database cluster.

If the source database cluster contains tablespaces that must be located in a different directory, additionally specify the `--tablespace-mapping` option:

```
pg_probackup3 catchup -b catchup_mode --destination-pgdata=path_to_local_dir --stream
 --tablespace-mapping=OLDDIR=NEWDIR
```

To run the `catchup` command on parallel threads, specify the number of threads with the `--threads` or `--num-write-threads` and `--num-validate-threads` options:

```
pg_probackup3 catchup -b catchup_mode --destination-pgdata=path_to_local_dir --stream
 --threads=num_threads
```

3.11. More Examples

All examples below assume the remote mode of operations via SSH. If you are planning to run backup and restore operation locally, skip the “Setup passwordless SSH connection” step and omit all `--remote-*` options.

Examples are based on Ubuntu 22.04, Postgres Pro 17, and pg_probackup3

- `backup` — Postgres Pro role used to connect to the Postgres Pro cluster.
- `backupdb` — database used to connect to the Postgres Pro cluster.
- `backup_host` — host with the backup catalog.
- `backup_user` — user on `backup_host` running all pg_probackup3 operations.
- `/mnt/backups` — directory on `backup_host` where the backup catalog is stored.
- `postgres_host` — host with the Postgres Pro cluster.
- `postgres` — user on `postgres_host` under which Postgres Pro cluster processes are running..
- `/var/lib/pgpro/std-17/data` — Postgres Pro data directory on `postgres_host`.

3.11.1. Minimal Setup

This scenario illustrates setting up standalone FULL and DELTA backups.

1. **Set up passwordless SSH connection from `backup_host` to `postgres_host`:**

   ```
   [backup_user@backup_host] ssh-copy-id postgres@postgres_host
   ```

2. **Configure your Postgres Pro cluster.**

   For security purposes, it is recommended to use a separate database for backup operations.

```
postgres=#
CREATE DATABASE backupdb;
```

Connect to the `backupdb` database, create the `probackup` role, and grant the following permissions to this role:

```
backupdb=#
BEGIN;
CREATE ROLE backup WITH LOGIN REPLICATION;
GRANT USAGE ON SCHEMA pg_catalog TO backup;
GRANT EXECUTE ON FUNCTION pg_catalog.current_setting(text) TO backup;
GRANT EXECUTE ON FUNCTION pg_catalog.set_config(text, text, boolean) TO backup;
GRANT EXECUTE ON FUNCTION pg_catalog.pg_is_in_recovery() TO backup;
GRANT EXECUTE ON FUNCTION pg_catalog.pg_backup_start(text, boolean, boolean) TO
 backup;
GRANT EXECUTE ON FUNCTION pg_catalog.pg_backup_stop(boolean, boolean) TO backup;
GRANT EXECUTE ON FUNCTION pg_catalog.pg_create_restore_point(text) TO backup;
GRANT EXECUTE ON FUNCTION pg_catalog.pg_switch_wal() TO backup;
GRANT EXECUTE ON FUNCTION pg_catalog.pg_last_wal_replay_lsn() TO backup;
GRANT EXECUTE ON FUNCTION pg_catalog.txid_current() TO backup;
GRANT EXECUTE ON FUNCTION pg_catalog.txid_current_snapshot() TO backup;
GRANT EXECUTE ON FUNCTION pg_catalog.txid_snapshot_xmax(txid_snapshot) TO backup;
GRANT EXECUTE ON FUNCTION pg_catalog.pg_control_checkpoint() TO backup;
COMMIT;
```

Add the pgpro_bindump plugin in the `postgresql.conf` file:

```
echo "shared_preload_libraries = 'pgpro_bindump'" >> "/var/lib/pgpro/std-17/data/
postgresql.conf"
echo "walsender_plugin_libraries = 'pgpro_bindump' " >> "/var/lib/pgpro/std-17/
data/postgresql.conf"
echo "wal_level = 'replica'" >> "/var/lib/pgpro/std-17/data/postgresql.conf"
```

3. **Initialize the backup catalog:**

```
[backup_user@backup_host]$ pg_probackup3 init -B /mnt/backups
2024-12-09 07:40:27.198881] [363926] [135107950659968] [info] Backup catalog '/mnt/
backups' successfully initialized
```

4. **Add instance `pg-17` to the backup catalog:**

```
[backup_user@backup_host]$ pg_probackup3 add-instance -B /mnt/backups --instance
 pg-17 --remote-host=postgres_host --remote-user=postgres -D var/lib/pgpro/std-17/
data
[2024-12-09 07:47:56.595727] [364390] [138813944502656] [info] Instance 'pg-17'
 successfully initialized
```

5. **Take a FULL backup:**

```
[backup_user@backup_host] pg_probackup3 backup -B /mnt/backups --instance pg-17
 -b FULL --stream --remote-host=postgres_host --remote-user=postgres -U backup -d
 backupdb --backup-id=1-full
[2024-12-09 23:44:49.602026] [425177] [123209585379712] [info] START BACKUP
 COMMAND= PGPRO_CALL_PLUGIN pgpro_bindump start_backup(LABEL '1-full');
[2024-12-09 23:44:49.645450] [425177] [123209585379712] [info] PG_PROBACKUP
 0/4000028 tli=1
[2024-12-09 23:44:49.652048] [425177] [123209585379712] [info] Created replication
 slot. Name='pg_probackup3_wal_streaming_425181', consistent point=0/0, snapshot
 name=, output plugin=
[2024-12-09 23:44:49.652185] [425177] [123209585379712] [info] BACKUP COMMAND
 PGPRO_CALL_PLUGIN pgpro_bindump copy_files(VERIFY_CHECKSUMS true, COMPRESS_ALG
 'none', COMPRESS_LVL 1);
```

```
[2024-12-09 23:44:49.652468] [425177] [123209573729984] [info] Starting new segment
 4
[2024-12-09 23:44:49.769640] [425177] [123209585379712] [info] BACKUP COMMAND
 PGPRO_CALL_PLUGIN pgpro_bindump stop_backup(STREAM true, COMPRESS_ALG 'none',
 COMPRESS_LVL 1);
[2024-12-09 23:44:49.805112] [425177] [123209573729984] [info] Stopping segment 4
[2024-12-09 23:44:49.805316] [425177] [123209573729984] [info] finished streaming
 WAL at 0/5000000 (timeline 1)
[2024-12-09 23:44:49.805343] [425177] [123209573729984] [info] WAL streaming: WAL
 streaming stop requested at 0/4000138, stopping at 0/5000000
[2024-12-09 23:44:49.805935] [425177] [123209585379712] [info] PG_PROBACKUP-STOP
 0/4000138 tli=1 bytes written=39430093 bytes compressed=39430093
[2024-12-09 23:44:49.806484] [425177] [123209585379712] [info] Backup time 206
[2024-12-09 23:44:49.806515] [425177] [123209585379712] [info] Backup 1-full
 completed successfully.
INFO: Backup 1-full completed successfully.
[2024-12-09 23:44:49.806592] [425177] [123209585379712] [info] Start validate  1-
full ...
[2024-12-09 23:44:49.807204] [425177] [123209585379712] [info] Validating backup 1-
full
[2024-12-09 23:44:49.912115] [425177] [123209585379712] [info] Validate time 104
[2024-12-09 23:44:49.912398] [425177] [123209585379712] [info] INFO: Backup 1-full
 is valid
```

6. **Let's take a look at the backup catalog:**

```
[backup_user@backup_host] pg_probackup3 show -B /mnt/backups --instance pg-17

BACKUP INSTANCE 'pg-17', version 3
============================================================================================
 Instance Version ID     End time                 Mode WAL Mode TLI Duration Data
 WAL Zalg Zratio Start LSN Stop LSN  Status
============================================================================================
pg-17      16      1-full 2024-12-09 23:44:49+0000 FULL STREAM   1            38MB
 -   none 1.00   0/4000028 0/4000138 OK
```

7. **Take an incremental backup in the DELTA mode:**

```
[backup_user@backup_host] pg_probackup3 backup -B /mnt/backups --instance pg-17 -
b delta --stream --remote-host=postgres_host --remote-user=postgres -U backup -d
 backupdb --parent-backup-id=1-full --backup-id=1-delta
[2024-12-10 01:00:50.804867] [430043] [130779551140224] [info] This PostgreSQL
 instance was initialized with data block checksums. Data block corruption will be
 detected
[2024-12-10 01:00:50.805233] [430043] [130779551140224] [info] START BACKUP
 COMMAND= PGPRO_CALL_PLUGIN pgpro_bindump start_backup(LABEL '1-delta', START_LSN
 '0/4000028');
[2024-12-10 01:00:50.843249] [430043] [130779551140224] [info] PG_PROBACKUP
 0/6000028 tli=1
[2024-12-10 01:00:50.850799] [430043] [130779551140224] [info] Created replication
 slot. Name='pg_probackup3_wal_streaming_430047', consistent point=0/0, snapshot
 name=, output plugin=
[2024-12-10 01:00:50.850898] [430043] [130779551140224] [info] BACKUP COMMAND
 PGPRO_CALL_PLUGIN pgpro_bindump copy_files(VERIFY_CHECKSUMS true, START_LSN
 '0/4000028', COMPRESS_ALG 'none', COMPRESS_LVL 1);
[2024-12-10 01:00:50.851124] [430043] [130779470366400] [info] Starting new segment
 6
[2024-12-10 01:00:50.877932] [430043] [130779551140224] [info] BACKUP COMMAND
 PGPRO_CALL_PLUGIN pgpro_bindump stop_backup(STREAM true, COMPRESS_ALG 'none',
 COMPRESS_LVL 1);
```

```
[2024-12-10 01:00:50.913070] [430043] [130779470366400] [info] Stopping segment 6
[2024-12-10 01:00:50.913284] [430043] [130779470366400] [info] finished streaming
 WAL at 0/7000000 (timeline 1)
[2024-12-10 01:00:50.913302] [430043] [130779470366400] [info] WAL streaming: WAL
 streaming stop requested at 0/6000138, stopping at 0/7000000
[2024-12-10 01:00:50.913497] [430043] [130779551140224] [info] PG_PROBACKUP-STOP
 0/6000138 tli=1 bytes written=786310 bytes compressed=786310
[2024-12-10 01:00:50.913868] [430043] [130779551140224] [info] Backup time 110
[2024-12-10 01:00:50.913884] [430043] [130779551140224] [info] Backup 1-delta
 completed successfully.
INFO: Backup 1-delta completed successfully.
[2024-12-10 01:00:50.913918] [430043] [130779551140224] [info] Start validate  1-
delta ...
[2024-12-10 01:00:50.914269] [430043] [130779551140224] [info] Validating backup 1-
delta
[2024-12-10 01:00:50.934892] [430043] [130779551140224] [info] Validate time 20
[2024-12-10 01:00:50.935188] [430043] [130779551140224] [info] INFO: Backup 1-delta
 is valid
```

8. **Let's add some parameters to pg_probackup3 configuration file, so that you can omit them from the command line:**

```
[backup_user@backup_host] pg_probackup3 set-config -B /mnt/backups --instance pg-17
 --remote-host=postgres_host --remote-user=postgres -U backup -d backupdb
[2024-12-10 01:03:18.173698] [430208] [125541616851328] [info] Instance 'pg-17'
 successfully updated
```

9. **Take another incremental backup in the DELTA mode, omitting some of the previous parameters:**

```
[backup_user@backup_host] pg_probackup3 backup -B /mnt/backups --instance pg-17 -b
 delta --stream --parent-backup-id=1-delta --backup-id=2-delta
[2024-12-10 01:26:33.325658] [431695] [135663496210816] [info] This PostgreSQL
 instance was initialized with data block checksums. Data block corruption will be
 detected
[2024-12-10 01:26:33.326140] [431695] [135663496210816] [info] START BACKUP
 COMMAND= PGPRO_CALL_PLUGIN pgpro_bindump start_backup(LABEL '2-delta', START_LSN
 '0/6000028');
[2024-12-10 01:26:33.365430] [431695] [135663496210816] [info] PG_PROBACKUP
 0/8000028 tli=1
[2024-12-10 01:26:33.372681] [431695] [135663496210816] [info] Created replication
 slot. Name='pg_probackup3_wal_streaming_431699', consistent point=0/0, snapshot
 name=, output plugin=
[2024-12-10 01:26:33.372762] [431695] [135663496210816] [info] BACKUP COMMAND
 PGPRO_CALL_PLUGIN pgpro_bindump copy_files(VERIFY_CHECKSUMS true, START_LSN
 '0/6000028', COMPRESS_ALG 'none', COMPRESS_LVL 1);
[2024-12-10 01:26:33.372966] [431695] [135663483619008] [info] Starting new segment
 8
[2024-12-10 01:26:33.407073] [431695] [135663496210816] [info] BACKUP COMMAND
 PGPRO_CALL_PLUGIN pgpro_bindump stop_backup(STREAM true, COMPRESS_ALG 'none',
 COMPRESS_LVL 1);
[2024-12-10 01:26:33.441125] [431695] [135663483619008] [info] Stopping segment 8
[2024-12-10 01:26:33.441303] [431695] [135663483619008] [info] finished streaming
 WAL at 0/9000000 (timeline 1)
[2024-12-10 01:26:33.441318] [431695] [135663483619008] [info] WAL streaming: WAL
 streaming stop requested at 0/8000138, stopping at 0/9000000
[2024-12-10 01:26:33.441497] [431695] [135663496210816] [info] PG_PROBACKUP-STOP
 0/8000138 tli=1 bytes written=786310 bytes compressed=786310
[2024-12-10 01:26:33.441809] [431695] [135663496210816] [info] Backup time 117
```

```
[2024-12-10 01:26:33.441822] [431695] [135663496210816] [info] Backup 2-delta
 completed successfully.
INFO: Backup 2-delta completed successfully.
[2024-12-10 01:26:33.441850] [431695] [135663496210816] [info] Start validate  2-
delta ...
[2024-12-10 01:26:33.442115] [431695] [135663496210816] [info] Validating backup 2-
delta
[2024-12-10 01:26:33.463554] [431695] [135663496210816] [info] Validate time 21
[2024-12-10 01:26:33.463728] [431695] [135663496210816] [info] INFO: Backup 2-delta
 is valid
```

10. **Let's take a look at the instance configuration:**

```
[backup_user@backup_host] pg_probackup3 show-config -B /mnt/backups --instance
 pg-17

# Backup instance information
system-identifier = 7446313657913924966
# Connection parameters
pguser = backup
pgdatabase = backupdb
pgdata = /var/lib/pgpro/std-17/data
# Logging parameters
log-level-console = info
log-level-file = off
log-format-console = plain
log-format-file = plain
log-filename = pg_probackup.log
log-rotation-size = 0
# Compression parameters
compress-algorithm = none
compress-level = 0
# Retention parameters
retention-redundancy = 0
retention-window = 0
wal-depth = 0
```

11. **Let's take a look at the backup catalog:**

```
[backup_user@backup_host] pg_probackup3 show -B /mnt/backups --instance pg-17
BACKUP INSTANCE 'pg-17', version 3
============================================================================================
 Instance Version ID      End time                 Mode  WAL Mode TLI Duration Data
  WAL Zalg Zratio Start LSN Stop LSN  Status
============================================================================================
pg-17      16      1-full  2024-12-09 23:44:49+0000 FULL  STREAM   1
 38MB  -   none 1.00   0/4000028 0/4000138 OK
pg-17      16      1-delta 2024-12-10 01:00:50+0000 DELTA STREAM   1
 768kB -   none 1.00   0/6000028 0/6000138 OK
pg-17      16      2-delta 2024-12-10 01:26:33+0000 DELTA STREAM   1
 768kB -   none 1.00   0/8000028 0/8000138 OK
```

Chapter 4. Reference

pg_probackup3

pg_probackup3 — utility to manage backup and recovery of Postgres Pro database clusters

Synopsis

pg_probackup3 add-instance -B *backup_dir* -D *data_dir* --instance *instance_name* --skip-if-exists

pg_probackup3 archive-get -B *backup_dir* --instance *instance_name* --wal-file-path *wal_file_path* --wal-file-name *wal_file_name* [*option*...]

pg_probackup3 archive-push -B *backup_dir* --instance *instance_name* --wal-file-path *wal_file_path* --wal-file-name *wal_file_name* [*option*...]

pg_probackup3 backup -B *backup_dir* --instance *instance_name* -b *backup_mode* [*option*...]

pg_probackup3 catchup -b *catchup_mode* --destination-pgdata=*path_to_local_dir* [*option*...]

pg_probackup3 del-instance -B *backup_dir* --instance *instance_name*

pg_probackup3 delete -B *backup_dir* --instance *instance_name* -i *backup_id*

pg_probackup3 file-map -B *backup_dir* --instance *instance_name* -i *backup_id* [*option*...]

pg_probackup3 fuse -B *backup_dir* --mnt-path *mnt_path* --instance *instance_name* -i *backup_id* --cache-swap-size *cache_swap_size* --cache-dir *cache_dir* [*option*...]

pg_probackup3 help [*command*]

pg_probackup3 init -B *backup_dir* --skip-if-exists

pg_probackup3 merge -B *backup_dir* --instance *instance_name* -i *backup_id* [*option*...]

pg_probackup3 restore -B *backup_dir* --instance *instance_name* [*option*...]

pg_probackup3 retention -B *backup_dir* --instance *instance_name* { --delete-wal | --delete-expired | --merge-expired } [*option*...]

pg_probackup3 send-backup -B *backup_dir* --instance *instance_name* -i *backup_id* -p *port* -h *host* [*option*...]

pg_probackup3 server-info [*option*...]

pg_probackup3 set-backup -B *backup_dir* --instance *instance_name* -i *backup_id* [*option*...]

pg_probackup3 set-config -B *backup_dir* --instance *instance_name* [*option*...]

pg_probackup3 show -B *backup_dir* [*option*...]

pg_probackup3 show-config -B *backup_dir* --instance *instance_name* [*option*...]

pg_probackup3 validate -B *backup_dir* [*option*...]

pg_probackup3 version

Command-Line Reference

Commands

This section describes pg_probackup3 commands. Optional parameters are enclosed in square brackets. For detailed parameter descriptions, see the section Options.

add-instance

```
pg_probackup3 add-instance -B backup_dir -D data_dir --instance=instance_name
[--skip-if-exists] [--wal-archive-dir=directory_path] [--help]
[s3_options] [ssh_options] [logging_options]
[connection_options] [compression_options] [retention_options]
[buffer_options]
```

Initializes a new backup instance inside the backup catalog *backup_dir* and generates the `pg_probackup3.conf` configuration file that controls pg_probackup3 settings for the cluster with the specified *data_dir* data directory. If the catalog was already initialized, you can ignore the error by specifying `--skip-if-exists`.

By default, the WAL archive is stored in *backup_dir*`/wal/`*instance_name*. Use the `--wal-archive-dir` option to specify a custom directory. The storage type (local filesystem, SFTP, or S3) must match the instance configuration. The directory will be created automatically if it does not exist.

For more details of the command settings, see sections Common Options and Adding a New Backup Instance.

archive-get

```
pg_probackup3 archive-get -B backup_dir --instance=instance_name --wal-file-
path=wal_file_path --wal-file-name=wal_file_name
[--help] [-j | --threads=num_threads] [--batch-size=batch_size] [--prefetch-dir=path]
 [--wal-archive-dir=directory_path]
[ssh_options] [logging_options] [s3_options] [buffer_options]
```

Copies WAL files from the corresponding subdirectory of the backup catalog to the cluster's write-ahead log location. This command is automatically set by pg_probackup3 as part of the `restore_command` when restoring backups using a WAL archive. You do not need to set it manually if you use local storage for backups or remote mode.

If you use S3 interface, to ensure that the Postgres Pro server has access to S3 storage to fetch WAL files during restore, you can specify the `--config-file` option that defines the S3 configuration file with appropriate configuration settings, as described in the section called “Common Options”.

By default, the WAL archive is stored in *backup_dir*`/wal/`*instance_name*. Use the `--wal-archive-dir` option to specify a custom directory. The storage type (local filesystem, SFTP, or S3) must match the instance configuration. The directory must already exist.

The Postgres Pro server requests WAL segments one at a time. To speed up recovery, you can specify the `--batch-size` option to copy WAL segments in batches of the specified size. If `--batch-size` is used, you can also specify the `-j/--threads` option to copy the batch of WAL segments on multiple threads. If `--batch-size` is not specified but `-j/--threads` is, `--batch-size` will default to the `-j/--threads` value.

For more details of the command settings, see sections Common Options, Archiving Options, and Compression Options.

archive-push

```
pg_probackup3 archive-push -B backup_dir --instance=instance_name
--wal-file-name=wal_file_name [--wal-file-path=wal_file_path]
[--help] [--no-sync] [--overwrite] [--wal-archive-dir=directory_path]
[--archive-timeout=wait_time]
[--compress-algorithm=compression_algorithm]
[--compress-level=compression_level]
[-j | --threads=num_threads] [--batch-size=batch_size]
[ssh_options] [logging_options]
[s3_options] [buffer_options]
```

Copies WAL files into the corresponding subdirectory of the backup catalog and validates the backup instance by *instance_name* and system-identifier. If parameters of the backup instance and the cluster do not match, this command fails with the following error message: Refuse to push WAL segment segment_name into archive. Instance parameters mismatch.

By default, the WAL archive is stored in *backup_dir*/wal/*instance_name*. Use the --wal-archive-dir option to specify a custom directory. The storage type (local filesystem, SFTP, or S3) must match the instance configuration. The directory must already exist.

If the files to be copied already exist in the backup catalog, pg_probackup3 computes and compares their checksums. If the checksums match, archive-push skips the corresponding file and returns a successful execution code. Otherwise, archive-push fails with an error.

Each file is copied to a temporary file with the .part suffix. If the temporary file already exists, pg_probackup3 will wait archive_timeout seconds before discarding it. After the copy is done, atomic rename is performed. This algorithm ensures that a failed archive-push will not stall continuous archiving and that concurrent archiving from multiple sources into a single WAL archive has no risk of archive corruption.

The Postgres Pro server requests WAL segments one at a time. To speed up archiving, you can specify the --batch-size option to copy WAL segments in batches of the specified size. If --batch-size is used, you can also specify the -j/--threads option to copy the batch of WAL segments on multiple threads. If --batch-size is not specified but -j/--threads is, --batch-size will default to the -j/--threads value.

WAL segments copied to the archive are synced to disk unless the --no-sync flag is used.

You can use archive-push in the *archive_command* Postgres Pro parameter to set up continuous WAL archiving.

For more details of the command settings, see sections Common Options, Archiving Options, and Compression Options.

backup

```
pg_probackup3 backup -B backup_dir --instance=instance_name -b backup_mode -
s backup_source -i backup_id
[--with-file-map] [--help] [--progress] [-j num_threads]
[--num-write-threads num_threads] [--num-validate-threads num_threads]
[--num-segments] [--create-slot] [--transfer-mode]
[--no-validate] [--skip-block-validation]
[--archive-timeout=wait_time] [--external-dirs=external_directory_path]
[--no-sync] [--note=backup_note]
[connection_options] [compression_options] [ssh_options]
[pinning_options] [logging_options] [s3_options] [buffer_options]
```

Creates a backup copy of the Postgres Pro instance.

-b *mode*
--backup-mode=*mode*

Specifies the backup mode to use. Possible values are: FULL, DELTA, and PTRACK.

-s *backup_source*
--backup-source=*backup_source*

Specifies the backup data source. Possible values are: DIRECT, BASE, and PRO.

--num-segments *num_segments*

Specifies the number of the backup segments during the backup creation or merge. Must be a positive integer.

> **Note**
>
> If the specified value exceeds the system limit for simultaneously open files, the process will fail with the error message “too many open files”.

`--num-write-threads` *num_threads*

Specifies the number of threads for copying files. Overrides the `j`/`--threads` option for file copying.

`--num-validate-threads` *num_threads*

Specifies the number of threads for the backup validation. Overrides the `j`/`--threads` option for the backup validation.

`-C`
`--smooth-checkpoint`

Spreads out the checkpoint over a period of time. By default, pg_probackup3 tries to complete the checkpoint as soon as possible.

`--stream`

Makes a STREAM backup, which includes all the necessary WAL files by streaming them from the database server via replication protocol.

`--backup-pg-log`

Includes the log directory into the backup. This directory usually contains log messages. By default, log directory is excluded.

`-E` *external_directory_path*
`--external-dirs=`*external_directory_path*

Includes the specified directory into the backup by recursively copying its contents into a separate subdirectory in the backup catalog. This option is useful to back up scripts, SQL dump files, and configuration files located outside of the data directory. If you would like to back up several external directories, separate their paths by a colon on Unix and a semicolon on Windows.

`--archive-timeout=`*wait_time*

Sets the timeout for WAL segment archiving and streaming, in seconds. By default, pg_probackup3 waits 300 seconds.

`--skip-block-validation`

Disables block-level checksum verification to speed up the backup process.

`--no-validate`

Skips automatic validation after the backup is taken. You can use this flag if you validate backups regularly and would like to save time when running backup operations.

It is recommended to use this flag when creating a backup to an S3 storage. Due to some features of S3 storages, automatic validation may appear incorrect in this case. Skip automatic validation and then perform validation using a separate validate command.

`--no-sync`

Do not sync backed up files to disk. You can use this flag to speed up the backup process. Using this flag can result in data corruption in case of operating system or hardware crash. If you use this option, it is recommended to run the validate command once the backup is complete to detect possible issues.

`--note=backup_note`

Sets the text note for backup copy. If `backup_note` contains newline characters, then only substring before first newline character will be saved. Max size of text note is 1 KB. The `'none'` value removes current note.

`--with-file-map`

Enables file map generation. Required for the fuse command.

`--transfer-mode=transfer_mode`

Specifies the method of sending data from a server to an application.

> **Note**
>
> This option is only available in the PRO backup data source mode.

Possible values:

- `raw` — unpacked data is sent in random blocks of arbitrary size.
- `packed` — packed data is sent in blocks of 128 KB with a common header.

`packed` is the default value (recommended).

For more details of the command settings, see sections Common Options, Connection Options, Pinning Options, SSH Options, Compression Options, and Logging Options.

For details on usage, see the section Creating a Backup.

catchup

```
pg_probackup3 catchup -b catchup_mode
--destination-pgdata=path_to_local_dir
[--temp-slot] [-P | --perm-slot] [-S | --slot=slot_name] [-j | --threads=num_threads]
[--num-write-threads num_threads] [--num-validate-threads num_threads]
[-x | --exclude-path=path_prefix]
[-T OLDDIR=NEWDIR] [connection_options] [logging_options] [compression_options]
```

Creates a copy of a Postgres Pro instance without using the backup catalog.

`-b catchup_mode`
`--backup-mode=catchup_mode`

Specifies the catchup mode to use. Possible values are: `FULL`, `DELTA`, and `PTRACK`.

`--destination-pgdata=path_to_local_dir`

Specifies the path to the local data directory to copy to.

`-j num_threads`
`--threads=num_threads`

Sets the number of parallel threads for `catchup` process.

`--num-write-threads num_threads`

Specifies the number of threads for copying files. Overrides the `j`/`--threads` option for file copying.

`--num-validate-threads num_threads`

Specifies the number of threads for the backup validation. Overrides the `j`/`--threads` option for the backup validation.

`--stream`

Copies the instance in the STREAM WAL delivery mode, including all the necessary WAL files by streaming them from the server via replication protocol.

`-x=`*`path_prefix`*
`--exclude-path=`*`path_prefix`*

Specifies a prefix for files to exclude from the synchronization of Postgres Pro instances during copying. The prefix must contain a path relative to the data directory of an instance. If the prefix specifies a directory, all files in this directory will not be synchronized.

> **Warning**
>
> This option is dangerous since excluding files from synchronization can result in incomplete synchronization; use with care.

`--temp-slot`

Creates a *temporary* physical replication slot for streaming WAL from the Postgres Pro instance being copied. It ensures that all the required WAL segments remain available if WAL is rotated while the backup is in progress. This flag cannot be used together with the `--perm-slot` flag. The default slot name is `pg_probackup3_wal_streaming_`*`backend_pid`*, where *`backend_pid`* is a PID of the Postgres Pro process. To change it, use the `--slot`/`-S` option and explicitly specify `--temp-slot`.

`-P`
`--perm-slot`

Creates a *permanent* physical replication slot for streaming WAL from the Postgres Pro instance being copied. This flag cannot be used together with the `--temp-slot` flag. The default slot name is `pg_probackup_perm_slot`, which can be changed using the `--slot`/`-S` option.

`-S` *`slot_name`*
`--slot=`*`slot_name`*

Specifies the replication slot to connect to for WAL streaming. This option can only be used together with the `--stream` flag.

`-T` *`OLDDIR=NEWDIR`*
`--tablespace-mapping=`*`OLDDIR=NEWDIR`*

Relocates the tablespace from the *`OLDDIR`* to the *`NEWDIR`* directory at the time of recovery. Both *`OLDDIR`* and *`NEWDIR`* must be absolute paths. If the path contains the equals sign (=), escape it with a backslash. This option can be specified multiple times for multiple tablespaces.

del-instance

`pg_probackup3 del-instance -B` *`backup_dir`* `--instance=`*`instance_name`* `[`*`s3_options`*`] [--help] [`*`ssh_options`*`] [`*`logging_options`*`] [`*`buffer_options`*`]`

Deletes all backups and WAL files associated with the specified instance.

For more details of the command settings, see the section Common Options.

delete

`pg_probackup3 delete -B` *`backup_dir`* `--instance=`*`instance_name`* `-i` *`backup_id`*
`[--help] [--progress] [--status=`*`backup_status`*`]`
`[--dry-run] [`*`logging_options`*`] [`*`ssh_options`*`]`
`[`*`s3_options`*`] [`*`buffer_options`*`]`

Deletes backups with specified *`backup_id`*.

`--dry-run`

Initiates a trial run of the `delete` command, which does not actually make any changes, that is, it does not delete files on disk. This flag allows you to check that all the command options are correct and the command is ready to run.

`--status`

Allows deleting all backups with a specific status.

For details, see the section Deleting Backups.

file-map

`pg_probackup3 file-map -B` *`backup_dir`* `--instance=`*`instance_name`* `-i` *`backup_id`*

Enables file map generation for an existing backup chain.

If file maps already exist for the specified backups, `file-map` overwrites them with newly generated versions.

fuse

`pg_probackup3 fuse -B` *`backup_dir`* `--mnt-path=`*`mnt_path`* `--instance=`*`instance_name`*
`-i` *`backup_id`* `[--cache-swap-size=`*`cache_swap_size`*`] [--cache-dir=`*`cache_dir`*`] [--detach] [--unmount] [--help]`
`[`*`ssh_options`*`] [`*`logging_options`*`] [`*`s3_options`*`] [`*`buffer_options`*`]`

Mounts a backup directory as a virtual file system and allows the Postgres Pro server to run on top of it. Refer to Section 3.7 for more details on the process.

`--cache-swap-size`

Specifies the amount of data (in MB) stored in memory. The default value is 128 MB. When the cache exceeds this size, changes are flushed to the nearby disk. This allows working with a database snapshot without modifying the actual backup. The cache is cleared when the Postgres Pro server is stopped.

`--cache-dir=`*`cache_dir`*

Specifies the path to the FUSE cache directory. If omitted, the system temporary directory is used.

`--detach`

Sets the FUSE filesystem to be run in the background (by default, it runs in the foreground). This is useful for automation.

`--unmount`

Unmounts the FUSE filesystem that was previously mounted with the `fuse` command. This option only requires the mount path (the `--mnt-path` option), which must be the same that was used for mounting.

help

`pg_probackup3 help [`*`command`*`]`

Displays the synopsis of pg_probackup3 commands. If one of the pg_probackup3 commands is specified, shows detailed information about the options that can be used with this command.

init

`pg_probackup3 init -B` *`backup_dir`* `[--skip-if-exists] [`*`s3_options`*`] [--help]`
`[`*`ssh_options`*`] [`*`logging_options`*`] [`*`buffer_options`*`]`

Initializes the backup catalog in *`backup_dir`* that will store backup copies, WAL archive, and meta information for the backed up database clusters. If the specified *`backup_dir`* already exists, it must be

empty. Otherwise, pg_probackup3 displays a corresponding error message. You can ignore this error by specifying the `--skip-if-exists` option. Although the backup will not be initialized, the application will return 0 code.

For more details of the process, refer to the section Initializing a Backup Catalog. For more details of the command settings, see the section Common Options.

merge

```
pg_probackup3 merge -B backup_dir --instance=instance_name -i backup_id --merge-from-
id=merge_from --merge-interval=merge_interval
[-t | --target-backup-id=backup_id] [-j num_threads] [--progress] [--no-validate] [--
no-sync]
[--with-file-map] [--keep-backups] [--dry-run] [--help] [logging_options] [ssh_options]
 [s3_options] [buffer_options]
```

Merges backups that belong to a common incremental backup chain. If you specify a full backup, it will be merged with its first incremental backup. If you specify an incremental backup, it will be merged to its parent full backup, together with all incremental backups between them. Once the merge is complete, the full backup takes in all the merged data, and the incremental backups are removed as redundant. You can also merge chains of incremental backups by specifying the first and the last incremental backup or the time interval (in hours) after the first backup.

`--no-validate`

Skips automatic validation before and after merge.

`--no-sync`

Do not sync merged files to disk. You can use this flag to speed up the merge process. Using this flag can result in data corruption in case of operating system or hardware crash.

`-t`
`--target-backup-id`

Specifies an ID of the merged backups.

`--keep-backups`

Preserves original backups after merging.

`--merge-from-id`

Specifies an ID of the first incremental backup from the backup chain for merge.

`--merge-interval`

Specifies a time period (in hours) before merging a chain of incremental backups.

`--with-file-map`

Enables file map generation. Required for the fuse command.

For more details of the command settings, see sections Common Options and Merging Backups.

restore

```
pg_probackup3 restore -B backup_dir --instance=instance_name
[--help] [-D data_dir] [-i backup_id] [--wal-archive-dir=directory_path]
[--progress] [-T OLDDIR=NEWDIR]
[--external-mapping=OLDDIR=NEWDIR] [--skip-external-dirs]
[-j num_threads] [--num-validate-threads num_threads]
[-R | --restore-as-replica] [--no-validate] [--skip-block-validation]
[--no-sync] [--restore-command=cmdline]
[--primary-conninfo=primary_conninfo]
```

[--primary-slot-name=*slot_name*]
[*recovery_target_options*] [*logging_options*]
[*ssh_options*] [*s3_options*] [*buffer_options*]

Restores the Postgres Pro instance from a backup located in the *backup_dir* backup catalog.

> **Note**
>
> While backup files for restore can be retrieved from different sources (the file system, S3, or SSH SFTP), pg_probackup3 can only restore the Postgres Pro server PGDATA to a local file system.

> **Note**
>
> The restore command does not support the --threads option yet. The number of threads will match the number of segments in the backup.

-R
--restore-as-replica

Creates a minimal recovery configuration file to facilitate setting up a standby server. If the replication connection requires a password, you must specify the password manually in the *primary_conninfo* parameter as it is not included. pg_probackup3 writes these settings into the probackup_recovery.conf file in the data directory and then includes them into the postgresql.auto.conf when the cluster is started.

--primary-conninfo=*primary_conninfo*

Sets the *primary_conninfo* parameter to the specified value. This option will be ignored unless the -R flag is specified.

Example: --primary-conninfo="host=192.168.1.50 port=5432 user=foo password=foopass"

--primary-slot-name=*slot_name*

Sets the *primary_slot_name* parameter to the specified value. This option will be ignored unless the -R flag is specified.

-T *OLDDIR*=*NEWDIR*
--tablespace-mapping=*OLDDIR*=*NEWDIR*

Relocates the tablespace from the *OLDDIR* to the *NEWDIR* directory at the time of recovery. Both *OLDDIR* and *NEWDIR* must be absolute paths. If the path contains the equals sign (=), escape it with a backslash. This option can be specified multiple times for multiple tablespaces.

--external-mapping=*OLDDIR*=*NEWDIR*

Relocates an external directory included into the backup from the *OLDDIR* to the *NEWDIR* directory at the time of recovery. Both *OLDDIR* and *NEWDIR* must be absolute paths. If the path contains the equals sign (=), escape it with a backslash. This option can be specified multiple times for multiple directories.

--skip-external-dirs

Skip external directories included into the backup with the --external-dirs option. The contents of these directories will not be restored.

--skip-block-validation

Disables block-level checksum verification to speed up validation. During automatic validation before the restore only file-level checksums will be verified.

`--no-validate`

Skips backup validation. You can use this flag if you validate backups regularly and would like to save time when running restore operations.

`--restore-command=`*`cmdline`*

Sets the *restore_command* parameter to the specified command. For example: `--restore-command='cp /mnt/server/archivedir/%f "%p"'`

`--wal-archive-dir=`*`directory_path`*

Specifies the directory for the WAL archive.

By default, the WAL archive is stored in *backup_dir*`/wal/`*instance_name*. Use the `--wal-archive-dir` option to specify a custom directory. The storage type (local filesystem, SFTP, or S3) must match the instance configuration. The directory must already exist.

`--no-sync`

Do not sync restored files to disk. You can use this flag to speed up restore process. Using this flag can result in data corruption in case of operating system or hardware crash. If it happens, you have to run the restore command again.

For more details of the command settings, see sections Common Options, Recovery Target Options, SSH Options, Remote WAL Archive Options, Logging Options.

For details on usage, see the section Restoring a Cluster.

retention

```
pg_probackup3 retention -B backup_dir --instance=instance_name
[--retention-redundancy] [--retention-window] [--dry-run] [--merge-expired] [--wal-
archive-dir=directory_path]
[--delete-expired] [--delete-wal] [pinning_options]
[ssh_options] [s3_options] [buffer_options]
```

Sets the backup retention policy for an instance or directory and launches backup merge or purge according to the specified parameters.

By default, the WAL archive is stored in *backup_dir*`/wal/`*instance_name*. Use the `--wal-archive-dir` option to specify a custom directory. The storage type (local filesystem, SFTP, or S3) must match the instance configuration. The directory must already exist.

For more details of the command settings, see the section Retention Options.

send-backup

```
pg_probackup3 send-backup -B backup_dir --instance=instance_name
-i backup_id -p port -h host [--no-merge]
```

Sends backup data to a remote system via the specified port using multithreading.

Runs on the local machine that contains the backup catalog.

Before the data transfer, the backup chain is merged into a temporary file, which is deleted after the transfer is complete. The merge can be disabled using the `--no-merge` flag — in this case, data will be transferred as-is.

See also the section Remote Restore.

server-info

```
pg_probackup3 server-info [--format=plain|json] [connection_options]
```

Displays availability of the backup options for the current Postgres Pro instance. Includes basic information about the instance and pg_probackup3 version. Example output is as follows:

```
CAN_USE_API = true
CAN_USE_CLI = true
CAN_USE_S3 = true
CAN_USE_CFS = true
CAN_USE_FUSE = true
CAN_USE_DIRECT = true
CAN_USE_BASE = true
CAN_USE_PRO = true
CAN_USE_MULTITHREAD = true
IS_PRO_MODE_CONFIGURED = true
IS_DELTA_AVAILABLE = true
IS_PTRACK_CONFIGURED = true
IS_PAGE_AVAILABLE = false
IS_WALSUM_AVAILABLE = false
IS_PGDATA_ACCESS = true
SERVER_CHECKSUM_ENABLED = true
PG_VERSION = 170006
PG_VERSION_STRING = "Postgres Pro (enterprise) 17.6.1 on aarch64-apple23.6.0, compiled
 by Homebrew clang version 21.1.2, 64-bit"
PG_EDITION_STRING = "enterprise"
PGDATA = "/Users/username/projects/postgrespro/tmp_install/data"
TIMELINE = 1
SYSTEM_ID = 75800766633445207
PG_MAX_WAL_SENDERS = 20
PG_BLOCK_SIZE = 8192
PG_BLOCKS_IN_SEGMENT = 131072
PG_WAL_BLOCK_SIZE = 8192
PG_WAL_SEGMENT_SIZE = 16777216
PG_TABLESPACE_MAP = "[]"
APP_VERSION = 3.2.0
```

By default, the output is shown as plain text. Specify the `--format=json` option to get the result in the JSON format.

The `CAN_USE_*` parameters reflect licensing limitations and show which features are enabled in the current Postgres Pro version.

set-backup

```
pg_probackup3 set-backup -B backup_dir --instance=instance_name -i backup_id
{--ttl=ttl | --expire-time=time}
[--note=backup_note] [ssh_options]
[s3_options] [--help] [logging_options] [buffer_options]
```

Sets the provided backup-specific settings into the `backup.control` configuration file, or modifies the previously defined values.

`--note=backup_note`

Sets the text note for backup copy. If *backup_note* contains newline characters, then only the substring before the first newline character will be saved. The maximum size of a text note is 1 KB. The *'none'* value removes the current note.

For more details of the command settings, see sections Common Options and Pinning Options.

set-config

```
pg_probackup3 set-config -B backup_dir --instance=instance_name
[--help] [--pgdata=pgdata-path] [--wal-archive-dir=directory_path]
[--retention-redundancy=redundancy][--retention-window=window]
[compression_options] [connection_options]
```

```
[--archive-timeout=wait_time] [--external-dirs=external_directory_path]
[logging_options] [ssh_options] [buffer_options]
```

Adds the specified connection, compression, retention, logging, and external directory settings into the `pg_probackup3.conf` configuration file, or modifies the previously defined values.

By default, the WAL archive is stored in *backup_dir*`/wal/`*instance_name*. Use the `--wal-archive-dir` option to specify a custom directory. The storage type (local filesystem, SFTP, or S3) must match the instance configuration. The directory must already exist.

For all available settings, see the Options section.

It is **not recommended** to edit `pg_probackup3.conf` manually.

show

```
pg_probackup3 show -B backup_dir
[--help] [--instance=instance_name [-i backup_id | --archive [--wal-archive-
dir=directory_path]]]
[--show-log] [--format=plain|json] [--no-color] [--format=plain|json|tree]
[s3_options] [ssh_options]
[logging_options] [buffer_options]
```

Shows the contents of the backup catalog. If *instance_name* and *backup_id* are specified, shows detailed information about this backup. If the `--archive` option is specified, shows the contents of the WAL archive, which is located in *backup_dir*`/wal/`*instance_name* by default. Use the `--wal-archive-dir` option to specify a custom directory.

By default, the contents of the backup catalog is shown as plain text. You can specify the `--format=json` option to get the result in the JSON format. If `--no-color` flag is used, then the output is not colored. You can also use the `--format=tree` option to see the list of backups as a tree.

For details on usage, see the sections Managing the Backup Catalog and Viewing WAL Archive Information.

show-config

```
pg_probackup3 show-config -B backup_dir --instance instance_name
[--format=plain|json] [s3_options] [ssh_options]
[logging_options] [buffer_options]
```

Displays all the current pg_probackup3 configuration settings, including those that are specified in the `pg_probackup3.conf` configuration file located in the *backup_dir*`/backups/`*instance_name* directory and those that were provided on a command line. The configuration settings are shown as plain text.

To edit `pg_probackup3.conf`, use the set-config command.

validate

```
pg_probackup3 validate -B backup_dir
[--help] [--instance=instance_name] [-i backup_id]
[-j num_threads] [--progress]
[--skip-block-validation] [buffer_options]
[logging_options] [ssh_options] [s3_options]
```

Verifies that all the files required to restore the cluster are present and are not corrupt. If you specify the *instance_name* without any additional options, pg_probackup3 validates all the backups available for this backup instance.

If the `--progress` option is specified, a list of the backup files and directories will be displayed during the validation process.

version

`pg_probackup3 version`

Prints pg_probackup3 version.

If `--format=json` is specified, the output is printed in the JSON format. This may be needed for native integration with JSON-based applications, such as PPEM. Example of a JSON output:

```
pg_probackup3 version
{
    "pg_probackup3":
    {
        "version": "3.0.0",
    },
    "compressions": [zlib, lz4, zstd]
}
```

Options

This section describes command-line options for pg_probackup3 commands. If the option value can be derived from an environment variable, this variable is specified below the command-line option, in the uppercase. Some values can be taken from the `pg_probackup3.conf` configuration file located in the backup catalog.

For details, see Section 2.4.

If an option is specified using more than one method, command-line input has the highest priority, while the `pg_probackup3.conf` settings have the lowest priority.

Common Options

The list of general options.

`--dry-run`

Initiates a trial run of the appropriate command, which does not actually do any changes, that is, it does not create, delete or move files on disk. This flag allows you to check that all the command options are correct and the command is ready to run. WAL streaming is skipped with `--dry-run`.

`-B` *directory*
`--backup-path=`*directory*
`BACKUP_PATH`

Specifies the absolute path to the backup catalog. Backup catalog is a directory where all backup files and meta information are stored. Since this option is required for most of the pg_probackup3 commands, you are recommended to specify it once in the `BACKUP_PATH` environment variable. In this case, you do not need to use this option each time on the command line.

`-D` *directory*
`--pgdata=`*directory*
`PGDATA`

Specifies the absolute path to the data directory of the database cluster. This option is mandatory only for the add-instance command. Other commands can take its value from the `PGDATA` environment variable, or from the `pg_probackup3.conf` configuration file.

`-i` *backup_id*
`--backup-id=`*backup_id*

Specifies the unique identifier of the backup.

`--parent-backup-id=`*parent_backup_id*

Specifies the unique identifier of the parent backup (used for incremental backups).

`--from-full`

Creates an incremental backup from the latest parent FULL backup.

`-j` *num_threads*
`--threads=`*num_threads*

Sets the number of parallel threads for `backup`, `restore`, `merge`, `validate`, and `archive-push` processes. Defaults to the number of processor cores.

`--num-validate-threads` *num_threads*

Sets the number of parallel threads during the backup validation, for example, when running the `backup` or `restore` command.

`--no-validate` disables `--num-validate-threads`.

`--progress`

Shows the progress of operations.

`--help`

Shows detailed information about the options that can be used with this command.

`-v` *version*
`--version=`*version*

Shows pg_probackup3 version.

`--config-file=`*file_name*

Specifies the S3 or SSH configuration file. Settings in the configuration file override the environment variables.

To generate the configuration file, run the set-config command with the `--config-file` option.

The generated file will include all explicitly specified S3 or SSH parameters, while passwords will be omitted and displayed as asterisks.

An example of the S3 configuration file:

```
access-key=admin
s3=on
s3-bucket=test
s3-host=127.0.0.1
s3-port=9000
s3-region=us-west-2
secret-key=***
```

If the configuration file contains both S3 and SSH options, S3 options will be used.

If the `--config-file` option is not specified, pg_probackup3 will first look for S3 and SSH configuration files at `/etc/pg_probackup/s3.config` or `/etc/pg_probackup/ssh.config` and then at `~postgres/.pg_probackup/s3.config` or `~postgres/.pg_probackup/ssh.config`, respectively.

Recovery Target Options

If continuous WAL archiving is configured, you can use one of these options with restore command to specify the moment up to which the database cluster must be restored.

`--recovery-target-stop=immediate|latest`

Defines when to stop the recovery:

- The `immediate` value stops the recovery after reaching the consistent state of the specified backup. This is the default behavior for STREAM backups.
- The `latest` value continues the recovery until all WAL segments available in the archive are applied. Setting this value of `--recovery-target` also sets `--recovery-target-timeline` to `latest`.

`--recovery-target-timeline=`*`timeline`*

Specifies a particular timeline to be used for recovery:

- `current` — the timeline of the specified backup, default.
- `latest` — the timeline of the latest available backup.
- A numeric value.

`--recovery-target-lsn=`*`lsn`*

Specifies the LSN of the write-ahead log location up to which recovery will proceed.

`--recovery-target-name=`*`recovery_target_name`*

Specifies a named savepoint up to which to restore the cluster.

`--recovery-target-time=`*`time`*`|current|latest`

Specifies the timestamp up to which recovery will proceed. If the time zone offset is not specified, the local time zone is used.

Example: `--recovery-target-time="2027-04-09 18:21:32+00"`

`--recovery-target-xid=`*`xid`*

Specifies the transaction ID up to which recovery will proceed.

`--recovery-target-inclusive=`*`boolean`*

Specifies whether to stop just after the specified recovery target (`true`), or just before the recovery target (`false`). This option can only be used together with `--recovery-target-time`, `--recovery-target-lsn` or `--recovery-target-xid` options. The default depends on the *recovery_target_inclusive* parameter.

`--recovery-target-action=pause|promote|shutdown`

Specifies the action (*recovery_target_action*) the server should take when the recovery target is reached.

Default: `pause`

Retention Options

These options are used with the retention command.

For details on configuring retention policy, see the section Configuring Retention Policy.

`--retention-redundancy=`*`redundancy`*

Specifies the number of full backup copies to keep in the data directory. Must be a non-negative integer. The zero value disables this setting.

Default: `0`

`--retention-window=`*`window`*

Specifies the number of days of recoverability. Must be a non-negative integer. The zero value disables this setting.

Default: `0`

`--delete-wal`

Deletes WAL files that are no longer required to restore the cluster from any of the existing backups.

`--delete-expired`

Deletes backups that do not conform to the retention policy defined in the `pg_probackup3.conf` configuration file.

`--merge-expired`

Merges the oldest incremental backup that satisfies the requirements of retention policy with its parent backups that have already expired.

Pinning Options

You can use these options together with backup, set-backup, and retention commands.

For details on backup pinning, see the section Backup Pinning.

`--ttl=`*`ttl`*

Specifies the amount of time the backup should be pinned. Must be a non-negative integer. The zero value unpins the already pinned backup. Supported units: ms, s, min, h, d (s by default).

Example: `--ttl=30d`

`--expire-time=`*`time`*

Specifies the timestamp up to which the backup will stay pinned. Must be an ISO-8601 complaint timestamp. If the time zone offset is not specified, the local time zone is used.

Example: `--expire-time="2027-04-09 18:21:32+00"`

Logging Options

You can use these options with any command.

`--no-color`

Disable coloring for console log messages of `warning` and `error` levels.

`--log-level-console=`*`log_level`*

Controls which message levels are sent to the console log. Valid values are `trace`, `debug`, `info`, `warning`, `error` and `off`. Each level includes all the levels that follow it. The later the level, the fewer messages are sent. The `off` level disables console logging.

Default: `info`

> **Note**
>
> All console log messages are going to stderr, so the output of show and show-config commands does not mingle with log messages.

`--log-level-file=`*`log_level`*

Controls which message levels are sent to a log file. Valid values are `trace`, `debug`, `info`, `warning`, `error`, and `off`. Each level includes all the levels that follow it. The later the level, the fewer messages are sent. The `off` level disables file logging.

Default: `off`

`--log-backup=log_level`

Controls which message levels are sent to a backup log file created in the backup directory when running the `backup` command. Valid values are `trace`, `debug`, `info`, `warning`, `error`, and `off`. Each level includes all the levels that follow it. The later the level, the fewer messages are sent. The `off` level disables file logging.

Default: `info`

`--log-filename=log_filename`

Defines the filenames of the created log files. The filenames are treated as a `strftime` pattern, so you can use %-escapes to specify time-varying filenames.

> **Note**
>
> Starting from PostgreSQL 17, stricter validation of percent-sign placeholders takes place in shell commands such as *`archive_command`*. In this context, `%%` should be used instead of single `%`. For more details, refer to *the corresponding PostgreSQL commit*.

Default: `pg_probackup.log`

For example, if you specify the `pg_probackup-%%u.log` pattern, pg_probackup3 generates a separate log file for each day of the week, with `%%u` replaced by the corresponding decimal number: `pg_probackup-1.log` for Monday, `pg_probackup-2.log` for Tuesday, and so on.

You can also use the file counter (`%%N`) and `strftime` format (pg_probackup-%%Y-%%m-%%d_%%H%%M%%S.log). The examples are shown in the table below.

Table 4.1. Filename Template Examples

Template	Expands to
file_%%N.log	file_1.log, file_2.log...
file_%%3N.log	file_001.log, file_002.log...
file_%%Y%%m%%d.log	file_20080705.log, file_20080706.log...
file_%%Y-%%m-%%d_%%H-%%M-%%S.%%N.log	file_2008-07-05_13-44-23.1.log, file_2008-07-06_16-00-10.2.log...

This option takes effect if file logging is enabled by the `--log-level-file` option.

`--error-log-filename=error_log_filename`

Defines the filenames of log files for error messages only. The filenames are treated as a `strftime` pattern, so you can use %-escapes to specify time-varying filenames.

> **Note**
>
> Starting from PostgreSQL 17, stricter validation of percent-sign placeholders takes place in shell commands such as *`archive_command`*. In this context, `%%` should be used instead of single `%`. For more details, refer to *the corresponding PostgreSQL commit*.

Default: none

For example, if you specify the `error-pg_probackup-%%u.log` pattern, pg_probackup3 generates a separate log file for each day of the week, with `%%u` replaced by the corresponding decimal number: `error-pg_probackup-1.log` for Monday, `error-pg_probackup-2.log` for Tuesday, and so on.

This option is useful for troubleshooting and monitoring.

`--log-directory=log_directory`

Defines the directory in which log files will be created. You must specify the absolute path. This directory is created lazily, when the first log message is written.

Note that the directory for log files is always created locally even if backups are created in the S3 storage. So be sure to pass a local path in `log_directory` when needed.

Default: `$BACKUP_PATH/log/`

`--log-format-console=log_format`

Defines the format of the console log. Only set from the command line. Note that you cannot specify this option in the `pg_probackup3.conf` configuration file through the set-config command and that the backup command also treats this option specified in the configuration file as an error. Possible values are:

- `plain` — sets the plain-text format of the console log.
- `json` — sets the JSON format of the console log.

Default: `plain`

`--log-format-file=log_format`

Defines the format of log files used. Possible values are:

- `plain` — sets the plain-text format of log files.
- `json` — sets the JSON format of log files.

Default: `plain`

`--log-rotation-size=log_rotation_size`

Defines the maximum size of an individual log file. If this value is reached, the log file is rotated once any pg_probackup3 command, except `help` or `version`, is launched. The zero value disables size-based rotation. Available unit values: B, kB, MB, GB, TB. If no unit is specified, the value defaults to bytes. Use file counter (%N) in the log filename pattern.

Default: `0`

Connection Options

You can use these options together with the backup command.

All *libpq environment variables* are supported.

`-d dbname`
`--pgdatabase=dbname`
`PGDATABASE`

Specifies the name of the database to connect to. The connection is used only for managing backup process, so you can connect to any existing database. If this option is not provided on the command line, `PGDATABASE` environment variable, or the `pg_probackup3.conf` configuration file, pg_probackup3 tries to take this value from the `PGUSER` environment variable, or from the current user name if `PGUSER` variable is not set.

`-h host`
`--pghost=host`
`PGHOST`

Specifies the host name of the system on which the server is running. If the value begins with a slash, it is used as a directory for the Unix domain socket.

Default: `localhost`

`-p port`
`--pgport=port`
`PGPORT`

Specifies the TCP port or the local Unix domain socket file extension on which the server is listening for connections.

Default: `5432`

`-U username`
`--pguser=username`
`PGUSER`

User name to connect as.

`-w`
`--no-password`

Disables a password prompt. If the server requires password authentication and a password is not available by other means such as a *.pgpass* file or `PGPASSWORD` environment variable, the connection attempt will fail. This flag can be useful in batch jobs and scripts where no user is present to enter a password.

`-W`
`--password`

Forces a password prompt. (Deprecated)

Compression Options

You can use these options together with backup and archive-push commands.

`--compress-algorithm=compression_algorithm`

Defines the algorithm to use for compressing data files. Possible values are `zlib`, `lz4`, `zstd`, and `none`. If set to any value but `none`, this option enables compression that uses the corresponding algorithm. Both data files and WAL files are compressed. By default, compression is disabled.

Default: `none`

> **Warning**
>
> Option value `lz4` for `--compress-algorithm` isn't currently supported in `archive-push` and `archive-get` commands.

`--compress-level=compression_level`

Defines the compression level.

> **Note**
>
> This option must be used together with the `--compress-algorithm` option.

Possible values depend on the compression algorithm specified:

- 0 — 9 for `zlib`
- 0 — 12 for `lz4`
- 0 — 22 for `zstd`

The value of 0 sets the default compression level for the specified algorithm:

- 6 for `zlib`
- 9 for `lz4`
- 3 for `zstd`

> **Note**
>
> The pure lz4 algorithm has only one compression level — 1. So, if the specified compression algorithm is `lz4` and `--compress-level` is greater than 1, the lz4hc algorithm is actually used, which is much slower although does better compression.

Default: `1`

Archiving Options

These options can be used with the archive-push command in the *archive_command* setting and the archive-get command in the *restore_command* setting.

Additionally, SSH options and logging options can be used.

`--wal-file-path=`*wal_file_path*

Provides the path to the WAL file in *archive_command* and *restore_command*. Use the `%p` variable as the value for this option or explicitly specify the path to a file outside of the data directory. If you skip this option, the path specified in `pg_probackup3.conf` will be used.

`--wal-file-name=`*wal_file_name*

Provides the name of the WAL file in *archive_command* and *restore_command*. Use the `%f` variable as the value for this option for correct processing. If the value of `--wal-file-path` is a path outside of the data directory, explicitly specify the filename.

`--overwrite`

Overwrites archived WAL file. Use this flag together with the archive-push command if the specified subdirectory of the backup catalog already contains this WAL file and it needs to be replaced with its newer copy. Otherwise, `archive-push` reports that a WAL segment already exists, and aborts the operation. If the file to replace has not changed, `archive-push` skips this file regardless of the `--overwrite` flag.

`--batch-size=`*batch_size*

Used to speed up archiving in case of `archive-push` or to speed up recovery in case of `archive-get`. Sets the maximum number of WAL files that can be copied into the archive by a single `archive-push` process or from the archive by a single `archive-get` process. If not specified, `--batch-size` defaults to the `-j/--threads` value.

`--archive-timeout=`*wait_time*

Sets the timeout for considering existing `.part` files to be stale. By default, pg_probackup3 waits 300 seconds. This option can be used only with the archive-push command.

`--no-sync`

Do not sync copied WAL files to disk. You can use this flag to speed up archiving process. Using this flag can result in WAL archive corruption in case of operating system or hardware crash. This option can be used only with archive-push command.

`--prefetch-dir=`*path*

Directory used to store prefetched WAL segments if `--batch-size` is used. Directory must be located on the same filesystem and on the same mountpoint the `PGDATA/pg_wal` is located. By default files

are stored in `PGDATA/pg_wal/pbk_prefetch` directory. This option can be used only with archive-get command.

Buffer Options

You can use these options with all commands.

`--buffer-size=`*size*

Specifies the buffer size for read and write operations. Must be a non-negative integer. The zero value disables this setting. The default value is 0.

`--buffer-read-size=`*size*

Specifies a separate buffer size for read operations. Must be a non-negative integer. The zero value disables this setting. The default value is 0.

`--buffer-write-size=`*size*

Specifies a separate extended buffer size for write operations. Must be a non-negative integer. The zero value disables this setting. The default value is 0.

You can explicitly specify units for any of the buffer options. Possible values: B, kB, MB, GB, TB. If no unit is specified, the value defaults to bytes.

S3 Options

This section describes the options needed to store backups in private clouds. These options can be used with any command that pg_probackup3 runs using S3 interface.

For more details, refer to the section Configuring S3 Connection section.

`--s3=`*s3_interface_provider*

Specifies the S3 interface provider. Possible values are:

- `minio` — MinIO object storage, compatible with S3 cloud storage service. With this provider, custom S3 server settings can be specified. The HTTP protocol, port 9000, and region `us-east-1` are used by default.
- `off` — explicitly disables S3 mode. This is the default value for `--s3` option.

With `--s3=minio`, pg_probackup3 will work fine for a VK Cloud storage if the S3 host address, port and protocol are properly specified (host address is `hb.vkcs.cloud` or the one specified in the appropriate section of the VK Cloud profile, port 443, and HTTPS protocol). Do not specify `--s3=minio` for the Amazon S3 storage.

`--s3-host=`*hostname*

Specifies the address of the S3 server. Can include the port number, separated by a colon. If the port number is not specified in a host string, the value of `--s3-port` is assumed. Do not add a colon if the port number is not specified.

`--s3-port=`*port_number*

Specifies the port of the S3 server.

`--s3-region=`*region*

Specifies the region of the S3 server. The default is `us-east-1`.

`--s3-bucket=`*bucket*

Specifies the name of the bucket on the S3 server.

`--access-key=`*access_key*

Specifies the access key from S3 storage.

`--secret-key=`*`password`*

Specifies the secret access key from S3 storage.

`--s3-secure=`*`protocol`*

Specifies the protocol to be used. Possible values:

- `ON` or `HTTPS` — HTTPS is used.
- `HTTP` — HTTP is used. This is the default value.

`--s3-retries=`*`retry_count`*

Sets the maximum number of attempts to execute an S3 request in case of failures. The default is `3`.

`--s3-timeout=`*`timeout`*

Sets the maximum amount of time to execute an HTTP request to the S3 server, in seconds. The default is `300`.

`--s3-ignore-cert-ver=ON|OFF`

Allows to skip the certificate host and peer verification. The default is `OFF`.

`--s3-ca-certificate=`*`ca_certificate`*

Specifies the path to the file with a trust Certificate Authority (CA) bundle.

`--s3-ca-path=`*`ca_path`*

Specifies the directory with trust CA certificates.

`--s3-client-cert=`*`client_cert`*

Sets the SSL client certificate.

`--s3-client-key=`*`client_key`*

Sets the private key file for the TLS and SSL client certificates.

`--s3-versioning=enabled|suspended|off`

Sets the S3 bucket support for object versioning. The default is `off`.

`--s3-http-compression=true|false`

Sets the "Accept-Encoding" HTTP header and decompresses received contents. The default is `false`.

The S3 performance options are described below.

`--s3-buffer-size=`*`size`* `[`*`unit`*`]`

Specifies the size of the read/write buffer for communicating with S3. You can explicitly specify units. Possible values: B, kB, MB, GB, TB. If no unit is specified, the value defaults to bytes.

> **Note**
>
> The `--s3-buffer-size` value must not be less than 5 MB. Smaller values will trigger a warning and will be automatically switched to `5MB`.

SSH Options

This section describes the options related to running pg_probackup3 operations remotely via SSH. These options can be used with all commands.

For details on configuring and using the remote mode via SSH, see Section 2.11 and Section 3.4.

`--remote-host=`*`destination`*

Specifies the remote host IP address or hostname to connect to.

`--remote-port=`*`port`*

Specifies the remote host port to connect to.

Default: `22`

`--remote-user=`*`username`*

Specifies remote host user for SSH connection. If you omit this option, the current user initiating the SSH connection is used.

`--remote-path=`*`path`*

Specifies pg_probackup3 installation directory on the remote system.

`--ssh-options=`*`ssh_options`*

Provides a string of SSH command-line options. For example, the following options can be used to set *`keep-alive`* for SSH connections opened by pg_probackup3: `--ssh-options="-o ServerAliveCountMax=5 -o ServerAliveInterval=60"`. For the full list of possible options, see *ssh_config manual page*.

`--ssh-password=`*`password`*

Specifies the password for SSH connection.

Remote WAL Archive Options

This section describes the options used to provide the arguments for the remote mode via SSH .

`--archive-host=`*`destination`*

Provides the argument for the `--remote-host` option in the `archive-get` command.

`--archive-port=`*`port`*

Provides the argument for the `--remote-port` option in the `archive-get` command.

Default: `22`

`--archive-user=`*`username`*

Provides the argument for the `--remote-user` option in the `archive-get` command. If you omit this option, the user that has started the Postgres Pro cluster is used.

Default: Postgres Pro user

Incremental Restore Options

This section describes the options for incremental cluster restore. These options can be used with the restore command.

`--I` *`incremental_mode`*
`--incremental-mode=`*`incremental_mode`*

Specifies the incremental mode to be used. Possible values are:

- `CHECKSUM` — replace only pages with mismatched checksum and LSN.
- `LSN` — replace only pages with LSN greater than point of divergence.
- `NONE` — regular restore.

Partial Backup and Restore Options

This section describes the options for partial cluster backup and restore. These options can be used with both the backup and restore commands:

`--db-exclude-oid=`*dboid*

Specifies the OID of the database to exclude from restore. All other databases in the cluster will be restored as usual, including `template0` and `template1`. This option can be specified multiple times for multiple databases.

`--db-include-oid=`*dboid*

Specifies the OID of the database to restore from a backup. All other databases in the cluster will not be restored, with the exception of `template0` and `template1`. This option can be specified multiple times for multiple databases.

These options can be used with the restore command:

`--db-exclude-name=`*dbname*

Specifies the name of the database to exclude from restore. All other databases in the cluster will be restored as usual, including `template0` and `template1`. This option can be specified multiple times for multiple databases.

`--db-include-name=`*dbname*

Specifies the name of the database to restore from a backup. All other databases in the cluster will not be restored, with the exception of `template0` and `template1`. This option can be specified multiple times for multiple databases.

> **Warning**
>
> Options `--db-exclude-oid` and `--db-include-oid` cannot be used together, as well as `--db-exclude-name` and `--db-include-name`.

Testing and Debugging Options

This section describes options useful only in a test or development environment.

`--cfs-nondatafile-mode`

Instructs backup command to backup CFS in a legacy mode. This allows fine-tuning compatibility with pg_probackup3. This option is mainly designed for testing.

`PGPROBACKUP_TESTS_SKIP_HIDDEN`

Instructs pg_probackup3 to ignore backups marked as hidden. Note that pg_probackup3 can never mark a backup as hidden. It can only be done by directly editing the `backup.control` file. This option can only be set with environment variables.

`PGPROBACKUP_TESTS_SKIP_EMPTY_COMMIT`

Instructs pg_probackup3 to skip empty commits after pg_backup_stop.

Authors

Postgres Professional, Moscow, Russia.

Appendix A. Release Notes

A.1. pg_probackup 3.2.1

Release date: 2025-02-19

This release is based on pg_probackup3 3.2.0 and provides the following improvements and bug fixes:

- Implemented arbitrary positioning support in compressed WAL files. WAL segments are now fully loaded into memory to enable efficient validation and other operations with compressed files.
- Fixed an issue that caused assertion error and crash when attempting to create a directory that already exists.
- Changed the `--batch-size` default behavior to fix multithreading issue. When unspecified, `--batch-size` now defaults to the `-j/--threads` value instead of the previous `1`.
- Optimized memory usage when creating file maps to prevent memory exhaustion for the `--with-file-map` and `file-map` operations.

A.2. pg_probackup 3.2.0

Release date: 2025-12-30

This release is based on pg_probackup3 3.1.1 and provides improvements to stability, performance, and compatibility of pg_probackup3. Notable changes are as follows:

- WAL and incremental backups:
  - Added WAL parsing.
  - Implemented waiting for complete WAL segment downloads.
  - Made WAL files missing from the last incremental backup to be ignored during restore to allow correct server startup.
  - Added WAL validation for the entire backup chain during merge.
  - Made `start_lsn` to be set according to the latest incremental backup after merge.
  - Limited the number of backup threads by available walsenders.
  - Made WAL files to be preserved only for the latest incremental backup during merge.
  - Added the `--wal-archive-dir` option that allows using a separate WAL archive directory.
  - Fixed logging levels for WAL archiving commands.
  - Fixed an error that occurred when specifying multiple `walsender_plugin_libraries`.
- Backup and restore:
  - Fixed compression level validation to support levels above 9 when the selected algorithm allows it.
  - Fixed handling of default tablespaces during incremental restore: they are no longer overwritten.
  - Fixed path generation in `restore_command`.
  - Added messages for backup validation completion and for the selected backup source when it is not explicitly specified.
  - Improved messages that are displayed when backup files are missing.
  - Added validation for files with duplicate IDs.

    - Fixed restoring from mixed incremental backup chains.
    - Simplified pre-restore validation: restoring from non-latest incremental backups no longer requires reading metadata of all subsequent incremental backups.
    - Fixed validation of the entire backup chain when merging multiple incremental backups into a single interval backup.
    - Added the `--db-include-oid` and `--db-exclude-oid` options for the `backup` and `restore` commands to backup and restore individual databases.
- `catchup`:
    - Implemented proper error handling and file skipping.
    - Added the `--exclude-path` option to exclude specific files and directories.
    - Added support for the `--threads`, `--temp-slot`, `--perm-slot`, `--slot-name`, and `--tablespace-mapping` options.
- CFS:
    - Implemented CFS file backup and restore support in the DIRECT mode.
    - Added CFS block validation with checksum verification.
    - Added a message indicating that CFS backup is not supported in the BASE mode.
- pgpro_backupstream (remote host restore):
    - Removed creation of `recovery.signal` in case of an error.
    - Fixed thread logic in `send-backup`: threads now terminate only after data has been completely read and processed by pgpro_backupstream.
- S3:
    - Made the bucket parameter mandatory.
    - Fixed compatibility with storage services that do not support versioned buckets.
    - Implemented reconnection with appropriate status messages upon connection loss.
    - Improved S3 write performance by selecting the least loaded segments.
    - Set a limit on the number of multipart uploads.
- FUSE:
    - Added the `--unmount` option for unmounting the FUSE filesystem.
    - Added the `--detach` option to run the process in the background.
    - Added support for the `chmod` and `chown` commands.
- TDE (Transparent Data Encryption):
    - Added TDE support.
    - Made the TDE status to be stored in backup metadata.

A.3. pg_probackup 3.1.1

Release date: 2025-10-28

This release is based on pg_probackup3 3.1.0 and provides improvements to stability, performance, and compatibility of pg_probackup3. Notable changes are as follows:

- Major improvements:

    - Implemented saving of history files during WAL streaming.
    - Fixed issues with remote restore.
- Other improvements:
    - Fixed failing of the `add-instance` command that could occur because handling of the `pg-pro_edition` configuration parameter depended on the localized error text.
    - Added verification of the system ID from the `pg_control` file before recovery from a backup done in the `DIRECT` data source mode by comparison with the `PGDATA` system ID. This prevents recovery from an inappropriate backup.
    - Added a warning after performing the `validate --instance` command if errors were detected during validation.
    - Fixed the interval merge logic and added the display of the parent backup ID for merged intervals.
    - Improved the exception handling in the work with the S3 storage. Added a safe cleanup of the resources.
    - Optimized handling of data in the CBOR (RFC 8949 Concise Binary Object Representation) format by using the buffer directly, without extra copying.
    - Fixed issues with backups in the `BASE` data source mode for PostgreSQL 14.
    - Improved handling of the `BACKUP_PATH` environment variable. When it is set, it is no longer needed to specify the backup directory through the `-B` option.

A.4. pg_probackup 3.1.0

Release date: 2025-09-25

This release is based on pg_probackup3 3.0.2 and provides new features, optimizations and bug fixes. Major changes are as follows:

- New features:
    - Added support for incremental restore via the `-I` | `--incremental-mode` option in the PRO mode.
    - Added the `--smooth-checkpoint` option for the `backup` command to spread the checkpoint operation over time, allowing processes to complete their work gracefully.
    - Added the `--batch-size` and `--threads` options for the `archive-push` command for multi-threaded WAL backup.
    - Added the pgpro_backupstream utility that allows performing restore to a remote host. Note that this utility is currently in *beta testing*. Some functionality, such as restoring from backups in the ARCHIVE mode and incremental restore, is not yet available.
- Bug fixes:
    - Fixed an error that occurred when taking a backup without specifying the instance and backup directory parameters.
    - Fixed the display of the STREAM and ARCHIVE WAL delivery modes.
    - Fixed an error that occurred when attempting to back up a cluster with CFS in the BASE mode if `shared_preload_libraries` was missing from the configuration file.
    - Fixed the display of backup IDs created using pg_probackup 2.X.
    - Fixed the list of available values for the `--log_rotation_size` option of the `set-config` command.
    - Fixed connection issues when using the `-W` | `--password` option.

- Other improvements:
  - Added support for databases without data checksums enabled for proper operation with Shardman.
  - Added validation for re-archiving existing WAL files. If a file with the same name exists, the new file is compared to the existing one. If the files differ, the behavior is controlled by the `--overwrite` option — the file is either overwritten or an error is thrown.
  - Added a restriction preventing the creation of backups with different `--backup-source` values (BASE, DIRECT, PRO) within a single backup chain.

A.5. pg_probackup 3.0.2

Release date: 2025-07-10

This release is based on pg_probackup3 3.0.0 and provides new features, optimizations and bug fixes. Major changes are as follows:

- FUSE:
  - Added support for working with uncompressed tablespaces.
  - Fixed FUSE operation with CFS.
  - Added the `file-map` command, which allows creating file maps for an existing backup chain starting from a selected incremental backup.
  - Fixed a FUSE error caused by insufficient disk space. Added the `--cache-dir` option to specify the FUSE cache directory path (uses OS temporary directory if not specified).
- S3:
  - Added options for managing HTTPS certificates.
  - Added the ability to set connection options via a configuration file using the `--config-file` option.
- Other improvements:
  - Added the `--status` option for the `delete` command, allowing deletion of backups with a specified status.
  - WARNING-level log messages are now highlighted in yellow. Added color highlighting for the `debug` and `trace`-level log messages.
  - Added the `--db-include` and `--db-exclude` options for the `restore` command. Databases can now be included and excluded both by OID (`--db-include-oid`/`--db-exclude-oid`) and by name.
  - Added support for specifying option values in size units: `B`, `KB`, `MB`, `GB`, `TB` (default is bytes).
  - Added the `--num-validate-threads` option to set a custom number of threads for validation.

A.6. pg_probackup 3.0.0

Release date: 2025-03-28

This is the first public release of pg_probackup3.

pg_probackup3 is based on *pg_probackup* where most of the functionality is implemented.

Major features are as follows:

- Version independence: The same pg_probackup3 version can now be used with different versions of Postgres Pro or PostgreSQL, ensuring compatibility and flexibility.

- API integration: pg_probackup3 can be integrated with various backup systems via API, thus offering centralized management of the backup process.
- Work without SSH: pg_probackup3 can work without an SSH connection, enabling more effective and secure data transfer.
- `FUSE`: pg_probackup3 introduces the new `fuse` command, which enables running a database instance directly from a backup without requiring a full restore, using the FUSE (Filesystem in User Space) mechanism.
- Operation by unprivileged users: pg_probackup3 can be started by users who do not have access rights to PGDATA. This helps to increase security and reduce the risk of potential errors.
- A new backup format: Each backup is now stored as a single file, making it easier to manage and store backups.
- pg_basebackup support: In the BASE data source mode, it is now possible to leverage the pg_basebackup replication protocol for improved backup speed and efficiency.
- PRO mode: pg_probackup3 introduces a proprietary replication protocol in the new PRO data source mode.
- Merging incremental backup chains: It is now possible to save disk space by merging chains of incremental backups.

Appendix B. Known Issues and Troubleshooting

B.1. Error `libpq.so.5: no version information available`

If you install PostgreSQL from the *PostgreSQL Yum Repository* RPM packages, you may encounter the following error when running pg_probackup3:

```
libpq.so.5: no version information available
```

These packages replace the system default version of the `libpq` library and may cause applications (not just pg_probackup3) to malfunction.

To resolve this problem, switch the system back to using the standard `libpq` version by running the following commands:

```
alternatives --install /etc/ld.so.conf.d/postgresql-pgdg-libs.conf pgsql-ld-conf /dev/
null 1320
alternatives --set pgsql-ld-conf /dev/null
ldconfig
```

Appendix C. Version Compatibility

pg_probackup3 follows *semantic* versioning.

The PRO backup data source mode requires the *pgpro_bindump* plugin (originally named contrib/pg_probackup3), available from the following Postgres Pro versions:

- Postgres Pro Standard: 14.18.1, 15.10.1, 16.6.1, 17.0.1, 17.2.1, 18.0.1
- Postgres Pro Enterprise: 14.18.1, 15.10.1, 16.6.1, 17.2.1, 18.0.1

pg_probackup3 supports the following Postgres Pro versions and editions:

Postgres Pro Version	**Corresponding pg_probackup3 Version**
15.10.1–15.12.1 16.6.1–16.8.1 17.2.1–17.4.1	3.0.0
14.18.1 15.13.1 16.9.1 17.5.1	3.0.1, 3.0.2
14.19.1 15.14.1 16.10.1 17.6.1 17.7.1	3.1.0, 3.1.1, 3.2.0

> **Note**
>
> Backward compatibility is maintained for all newly released versions, so new versions of pg_probackup3 remain compatible with previous Postgres Pro versions.

Index

C

P