pg_probackup 3.0.0 Documentation

pg_probackup 3.0.0 Documentation

Table of Contents

Chapter 1. About pg_probackup3

Table of Contents

pg_probackup3 is designed to manage backup and recovery of PostgreSQL database clusters. It supports Postgres Pro and PostgreSQL 15 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.

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 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, available exclusively in Postgres Pro Enterprise.
- Merging incremental backup chains: It is now possible to save disk space by merging chains of incremental backups.
- Completely reengineered core
- Redesigned architecture
- Improved performance

> **Note**
>
> Certain pg_probackup3 functionalities rely on features specific to Postgres Pro Enterprise. These functionalities are automatically turned off for pg_probackup3 included in Postgres Pro Standard.

Installing pg_probackup3

pg_probackup3 includes the following packages:

- `libpgprobackup3` — contains a shared library providing the API for backup creation, as well as `libpb3_encoder.so`, a dynamic library to be loaded by the corresponding server extension.
- `pg_probackup3` — contains the command-line utility for managing backups.

> **Note**
>
> All packages have to be installed and uninstalled together, as `pg_probackup3` and `libpg-probackup3` only support the Postgres Pro version 15 and higher.

> **Note**
>
> Before installing pg_probackup3, make sure you have installed the pgpro_bindump module.

To install pg_probackup3, follow the steps below.

1. **Download the archive with packages**

   Download the archive with packages using the provided link. The archive contains test repositories for building pg_probackup3 for the following operating systems:

   - Debian 12
   - Astra Linux Smolensk 1.8
   - Ubuntu 22.04/24.04
   - RHEL 9

2. **Extract the archive**

   Extract the archive using the following command:

   ```
   tar -xvzf pbk3.tar.gz pbk3/
   ```

3. **Configure the repository**

   1. Install the GPG key.

      The archive includes the `keys` directory containing the GPG key for connecting the repository. Install the key based on your system.

      - For Debian-based systems:

        ```
        sed -n '/^$/,/=$/p' "/path/to/pbk3/keys/GPG-KEY-
        POSTGRESPRO" | base64 -d | sudo tee "/etc/apt/
        trusted.gpg.d/postgrespro.gpg" > /dev/null
        ```

      - For RHEL-based systems:

```
sudo rpm --import /path/to/pbk3/keys/GPG-KEY-POSTGRESPRO
```

2. Connect the repository.

   - For Debian-based systems, create the file `/etc/apt/sources.list.d/pbk3.list` with the following content.

     - For Ubuntu 24.04:

       ```
       deb file:///path/to/pbk3/ubuntu noble main
       ```

     - For Ubuntu 22.04:

       ```
       deb file:///path/to/pbk3/ubuntu jammy main
       ```

   - For RHEL-based systems, create the file `/etc/yum.repos.d/pbk3.repo` with the following content:

     ```
     [pbk3]
     name=pbk3
     baseurl=file:///path/to/pbk3/rhel/9Server/os/x86_64/rpms
     enabled=1
     gpgcheck=1
     ```

4. **Install the packages**

   - For Debian-based systems, after connecting the repository, update the package list:

     ```
     sudo apt update
     ```

     Then install the required packages:

     ```
     sudo apt install libpgprobackup3 pg-probackup3
     ```

   - For RHEL-based Systems, install the packages using `yum`:

     ```
     sudo yum install libpgprobackup3 pg-probackup3
     ```

     Or `dnf`:

     ```
     sudo dnf install libpgprobackup3 pg-probackup3
     ```

5. **Verify the installation**

   Ensure that the packages are installed correctly.

   1. Check the Postgres Pro version:

      ```
      /opt/pgpro/ent-16/bin/postgres --version
      ```

   2. Check the pg_probackup3 version:

      ```
      pg_probackup3 --version
      ```

6. **Configure Postgres Pro for pg_probackup3**

   To enable pg_probackup3, add the following parameters in the `postgresql.conf` file:

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

Once the installation is complete, it is required to restart the Postgres Pro instance.

Versioning

pg_probackup3 follows semantic versioning.

Chapter 2. About pg_probackup3 Backup and Recovery

Table of Contents

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.

Features

As compared to other backup solutions, pg_probackup3 offers the following benefits that can help you implement different backup strategies and deal with large amounts of data:

- S3 support for storing data in private clouds using MinIO object storage, Amazon S3 storage, VK Cloud storage and Google Cloud Storage: provided in Postgres Pro Enterprise. Backup data is transferred to and from S3 without saving it in intermediate locations thus eliminating the need of having a large temporary storage.
- Tape ready: pg_probackup3 supports working with tape storage backup systems.
- NFS v4 and v5 support: pg_probackup3 allows storing backups in the network file system.
- Incremental backup: With three different incremental modes, you can plan the backup strategy in accordance with your data flow. Incremental backups allow you to save disk space and speed up backup as compared to taking full backups. It is also faster to restore the cluster by applying incremental backups than by replaying WAL files.
- 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.
- Backup catalog: getting the list of backups and the corresponding meta information in plain text or JSON formats.
- Archive catalog: getting the list of all WAL timelines and the corresponding meta information in plain text or JSON formats.
- Integration with other applications enabled by the API provided by the libprobackup library.

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. You can store backups for different instances in separate subdirectories of a single backup catalog.

Using pg_probackup3, you can take full or incremental backups:

- FULL backups contain all the data files required to restore the database cluster.
- Incremental backups 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.

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 the following *WAL delivery mode*:

- 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.

In pg_probackup3 there are the following modes of backup data sources:

- DIRECT. Does not use any replication protocol.
- BASE. Uses the pg_basebackup protocol.
- PRO. The default mode that uses the pg_probackup3 protocol. It is included in Postgres Pro Enterprise.

Limitations

pg_probackup3 currently has the following limitations:

- 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 15 or higher.

Chapter 3. Backup and Recovery Setup

Table of Contents

Once you have pg_probackup3 installed, complete the following setup:

- Initialize the backup catalog.
- Add a new backup instance to the backup catalog.
- Configure the database cluster to enable pg_probackup3 backups.
- Optionally, configure SSH for running pg_probackup3 operations in the remote mode.
- Optionally, configure S3 for running pg_probackup3 connected to the S3 storage.

Initializing the Backup Catalog

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

Before initializing the backup catalog, make sure the following prerequisites are fulfilled:

- pg_probackup3 is connected to the Postgres Pro server.
- The user launching pg_probackup3 has full access to the *`backup_dir`* directory.

To initialize the backup catalog, run the following command:

```
pg_probackup3 init -B backup_dir
```

where *`backup_dir`* is the path to the backup catalog. If the *`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 the backup catalog is initialized, you can add a new backup instance.

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 instance to the backup catalog for each 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 [remote_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.
- remote_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 remote_options, the specified parameters will be added to pg_probackup3.conf.

For details on how to fine-tune pg_probackup3 configuration, see the section called “Configuring pg_probackup3”.

The user launching 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 feature, 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.

Configuring pg_probackup3

Once the 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.

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.

> **Note**
>
> It is **not recommended** to edit pg_probackup3.conf manually.

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]
 [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.

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.

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;
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)
 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, since pg_probackup3 uses a replication protocol (the PRO or BASE modes) to retrieve file data or performs direct backups (the DIRECT mode). The PRO mode is set as 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 the Remote Mode, and Setting up PTRACK Backups.

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

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
 [remote_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 remote_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 the section called “Viewing WAL Archive Information”.

> **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/`*`instance_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.

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.

pg_probackup3 includes two applications for creating backups in the PTRACK mode:

- The PTRACK application for creating backups in the DIRECT mode.
- The pb3_ptrack application for creating backups in the PRO mode.

If you are going to use PTRACK backups in the DIRECT mode, complete the following additional steps.

> **Note**
>
> The permissions required for the role that will perform PTRACK backups (the `backup` role in the examples below) are listed in the section called “Configuring the Database Cluster”. 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;
   ```

To create PTRACK backups in the PRO mode, set the `pb3_ptrack.map_size` parameter to a positive integer in the `postgresql.conf` file and restart the server.

For optimal performance, it is recommended to set `pb3_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, pb3_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 `pb3_ptrack.map_size` to a higher value does not affect pb3_ptrack operation, but it is not recommended to set this parameter to a value higher than 1024.

> **Note**
>
> If you change the `pb3_ptrack.map_size` parameter value, the previously created pb3_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 `pb3_ptrack.map_size`.

> **Note**
>
> The pgpro_bindump module must be enabled before setting up pbk3_ptrack.

> **Warning**
>
> Enabling both PTRACK and pb3_ptrack applications at the same time will lead to critical errors and backup failure. Make sure only the required application is activated.

Configuring the Remote Mode

pg_probackup3 supports the remote mode that allows you to perform backup and WAL archiving operations remotely.

Set up SSH

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 Remote Mode 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.

Configuring S3 Connectivity

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.

An example configuration with a remote agent and a cloud storage (S3) is shown in Figure 3.1.

Figure 3.1. pg_probackup3 setup with a remote agent and S3

In this figure, the following logical components are shown:

Backup server

A server where the main process of pg_probackup3 runs and where local backups are stored.

Database server

A server with a database instance that needs to be backed up.

Remote agent

A secondary pg_probackup3 process running on the database server. Only applicable to the remote mode.

Cloud storage

A cloud storage for backups.

Set up Access to S3 Storage

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 S3 configuration file (see the `--s3-config-file` option in the section S3 Options for details).

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 or list of addresses of the S3 server. A list of one or several semicolon-delimited addresses. Do not add a semicolon after the last address in the list. Each address can include the port number, separated by a colon. If the port number is not specified, 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;10.4.13.56:443;172.17.0.1"
```

In this example, for the “127.0.0.1” address, the port 9000 is explicitly specified, for “10.4.13.56”, the port 443 is specified, while for the “172.17.0.1” address, port 80, specified through `PG_PROBACKUP_S3_PORT`, will be used.

If any of the specified addresses gets unavailable while pg_probackup3 is in operation, requests to the S3 storage are distribited between the rest of the specified addresses. That is, when several addresses are specified, pg_probackup3 performs load balancing of S3 requests.

`PG_PROBACKUP_S3_PORT`

The port of the S3 server.

`PG_PROBACKUP_S3_REGION`

The region of the S3 server.

`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 `ON`.

`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.

Chapter 4. Usage

Table of Contents

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

> **Note**
>
> BASE and DIRECT backup data source modes support 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.

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.

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 [--temp-slot]
```

The optional `--temp-slot` flag ensures that the required segments remain available if the WAL is rotated before the backup is complete.

> **Note**
>
> While `--temp-slot` is optional, it can still affect the success of the backup.

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.

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.

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. This ensures that the backup remains unchanged, and all operations are *read-only*.

Figure 4.1. 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**
>
> CFS (Compressed File System) and tablespaces are not currently supported.

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

Restoring a Cluster

> **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.

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 the section called “Performing Point-in-Time (PITR) Recovery”.

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
```

Partial Restore

You can restore particular databases without any special preparations using partial restore options with the restore command and OIDs of these databases.

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
```

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.

Performing Point-in-Time (PITR) Recovery

If you have enabled continuous WAL archiving before taking backups, you can restore the cluster to its state at an arbitrary point in time (recovery target) using recovery target options with the restore command.

You can use both STREAM and ARCHIVE backups for point-in-time recovery as long as the WAL archive is available at least starting from the time the backup was taken.

- To restore the cluster state at the exact time, specify the `--recovery-target-time` option, in the timestamp format. For example:

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

- To restore the current or latest cluster state, set the `--recovery-target-time` option value to `current` or `latest`, respectively:

  ```
  pg_probackup3 restore -B backup_dir --instance=instance_name --recovery-target-time="latest"
  ```

- To restore the cluster state up to a specific transaction ID, use the `--recovery-target-xid` option:

  ```
  pg_probackup3 restore -B backup_dir --instance=instance_name --recovery-target-xid=687
  ```

- To restore the cluster state up to the specific LSN, use `--recovery-target-lsn` option:

```
pg_probackup3 restore -B backup_dir --instance=instance_name --
recovery-target-lsn=16/B374D848
```

- To restore the cluster state up to the specific named restore point, use `--recovery-target-name` option:

```
pg_probackup3 restore -B backup_dir --instance=instance_name --
recovery-target-name="before_app_upgrade"
```

- To restore the backup to the latest state available in the WAL archive, use `--recovery-target-stop` option with `latest` value:

```
pg_probackup3 restore -B backup_dir --instance=instance_name --
recovery-target-stop="latest"
```

- To restore the cluster to the earliest point of consistency, use `--recovery-target-stop` option with the `immediate` value:

```
          pg_probackup3 restore -B backup_dir --
instance=instance_name --recovery-target-stop='immediate'
```

Managing the Backup Catalog

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

- View backup information
- View WAL Archive Information
- Merge backups
- Delete backups

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.
- `pgdata-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 `pgdata-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"
            }
        ]
    }
]
```

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.

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, 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.

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.

Using pg_probackup3 in the Remote Mode

pg_probackup3 supports the remote mode that allows you to perform `backup` operations remotely via SSH. In this mode, the backup catalog is stored on a local system, while Postgres Pro instance to be backed up is located on a remote system. You must have pg_probackup3 installed on both systems.

> **Note**
>
> pg_probackup3 relies on passwordless SSH connection for communication between the hosts.

> **Note**
>
> In addition to SSH connection, pg_probackup3 uses a regular connection to the database to manage the remote operation. See the section Configuring the Database Cluster for details on how to set up a database connection.

The typical workflow is as follows:

- On your backup host, configure pg_probackup3 as explained in the section Backup and Recovery Setup. For the add-instance and set-config commands, make sure to specify remote mode options that point to the database host with the Postgres Pro instance.
- If you would like to rely on ARCHIVE WAL delivery mode, configure continuous WAL archiving from the database host to the backup host as explained in the section Setting up continuous WAL archiving. For the archive-push and archive-get commands, you must specify the remote mode options that point to the backup host with the backup catalog.
- Run the backup command with remote mode options **on the backup host**. pg_probackup3 connects to the remote system via SSH and creates a backup locally.

For example, to create an 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 via the SSH connection through the port `2302`, run:

```
pg_probackup3 backup -B backup_dir --instance=instance_name -b
 FULL --remote-user=postgres --remote-host=192.168.0.2 --remote-
port=2302
```

Running pg_probackup3 on Parallel Threads

backup, restore, merge, delete, 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` and `--num-write-threads` command-line options. These options must be non-negative integers.

If `--threads` is 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.

If `--num-write-threads` is not specified, the number of write threads will match the number of read threads.

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.

> **Important**
>
> A technique is implemented that prevents repeatable copying of one file when pg_probackup3 runs on multiple threads. With this technique, however, when disks are slow or the system is overloaded, parallel copying might fail. To handle this situation, resolve issues with your system resources.

Checking Data Integrity

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.

Validating a Backup

pg_probackup3 calculates checksums for each file in a backup during the backup process. The process of checking checksums of backup data files is called *the backup validation*. By default, validation is run immediately after the backup is taken and right before the restore, to detect possible backup corruption.

> **Note**
>
> The backup validation includes checking checksums for CFS files.

If you would like to skip backup validation, you can specify the `--no-validate` flag when running backup and restore commands.

For example, to check that you can restore the database cluster from a backup copy up to transaction ID 4242, run this command:

```
pg_probackup3 validate -B backup_dir --instance=instance_name --
recovery-target-xid=4242
```

If validation completes successfully, pg_probackup3 displays the corresponding message. If validation fails, you will receive an error message with the exact time, transaction ID, and LSN up to which the recovery is possible.

If you specify *backup_id* via `-i/--backup-id` option, then only the backup copy with specified backup ID will be validated. If *backup_id* is specified with recovery target options, the `validate` command will check whether it is possible to restore the specified backup to the specified recovery target.

For example, to check that you can restore the database cluster from a backup copy with the `SBOL6P` backup ID up to the specified timestamp, run this command:

```
pg_probackup3 validate -B backup_dir --instance=instance_name -i
 SBOL6P --recovery-target-time="2024-04-10 18:18:26+03"
```

If you specify the *`backup_id`* of an incremental backup, all its parents starting from FULL backup will be validated.

If you omit all the parameters, all backups are validated.

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.

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.

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.

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
```

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`.

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_start_backup(text,
    boolean, boolean) TO backup;
   GRANT EXECUTE ON FUNCTION pg_catalog.pg_stop_backup(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 pg_probackup3 module pgpro_bindump 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 5. Reference

Table of Contents

Name

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

Synopsis

`pg_probackup3 version`

`pg_probackup3 help` [*command*]

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

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

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

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

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

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

`pg_probackup3 show -B` *backup_dir* [*option*…]

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

`pg_probackup3 restore -B` *backup_dir* `--instance` *instance_name* [*option*…]

`pg_probackup3 validate -B` *backup_dir* [*option*…]

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

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

`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 fuse -B` *backup_dir* `--mnt-path` *mnt_path* `--instance` *instance_name* `-i` *backup_id* `--cache-swap-size` *cache_swap_size* [*option*…]

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

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.

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]
}
```

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

add-instance

```
pg_probackup3 add-instance -B backup_dir -D data_dir --
instance=instance_name
[--skip-if-exists] [s3_options] [ssh_options] [--help]
 [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`.

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

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.

set-config

```
pg_probackup3 set-config -B backup_dir --instance=instance_name
[--help] [--pgdata=pgdata-path]
[--retention-redundancy=redundancy][--retention-window=window]
[compress_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.

For all available settings, see the Options section.

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

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.

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.

show

```
pg_probackup3 show -B backup_dir
[--help] [--instance=instance_name [-i backup_id | --archive]]
[--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 WAL archive of the backup catalog.

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.

backup

```
pg_probackup3 backup -B backup_dir --instance=instance_name -
b backup_mode -s backup_source -i backup_id
[--with-file-map] [--help] [-j num_threads] [--progress]
[--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”.

`--backup-threads` *num_threads*

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

`--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.

`--temp-slot[=true|false|on|off]`

Creates a *temporary* physical replication slot for streaming WAL from the backed up Postgres Pro instance. `--temp-slot` is enabled by default. It ensures that all the required WAL segments remain available if WAL is rotated while the backup is in progress. This flag can only be used together with the `--stream` flag. The default slot name is `pg_probackup_slot`. To change it, use the `--slot`/`-S` option and explicitly specify `--temp-slot` or `--temp-slot=true|on`.

`-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.

`--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.

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

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

restore

```
pg_probackup3 restore -B backup_dir --instance=instance_name
[--help] [-D data_dir] [-i backup_id]
[--progress] [-T OLDDIR=NEWDIR]
[--external-mapping=OLDDIR=NEWDIR] [--skip-external-dirs]
[--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.

`--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"'`

`--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, Remote Mode Options, Remote WAL Archive Options, Logging Options.

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

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.

For details, see the section Validating a Backup.

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.

delete

```
pg_probackup3 delete -B backup_dir --instance=instance_name
[--help] [--progress]
[--dry-run] [--no-sync] [logging_options] [ssh_options]
[s3_options] [buffer_options]
```

Deletes backups with specified *backup_id*.

`--no-sync`

Do not sync deleted files to disk. Using this flag can result in data corruption in case of operating system or hardware crash.

For details, see the section Deleting Backups.

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] [--compress]
[--archive-timeout=wait_time]
[--compress-algorithm=compression_algorithm]
[--compress-level=compression_level]
[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.`

If the files to be copied already exists 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.

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.

fuse

```
pg_probackup3 fuse -B backup_dir --mnt-path=mnt_path --
instance=instance_name
-i backup_id [--cache-swap-size=cache_swap_size] [--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.

`--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.

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`.

> **Note**
>
> The backup chain to be mounted must include the `--with-file-map` option. This option is available during the backup and merge operations.

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] [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 `--s3-config-file` option that defines the S3 configuration file with appropriate configuration settings, as described in the section called “S3 Options”.

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

retention

```
pg_probackup3 retention -B backup_dir --instance=instance_name
[--retention-redundancy] [--retention-window] [--dry-run] [--merge-
expired]
[--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.

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

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 the section called “Configuring pg_probackup3”.

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.

`--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.

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 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 `verbose`, `log`, `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 `verbose`, `log`, `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-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.

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.

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.

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`*

Maximum size of an individual log file. If this value is reached, the log file is rotated once a pg_probackup3 command is launched, except `help` and `version` commands. The zero value disables size-based rotation. Supported units: kB, MB, GB, TB (kB by default).

Default: `0`

`--log-rotation-age=`*`log_rotation_age`*

Maximum lifetime of an individual log file. If this value is reached, the log file is rotated once a pg_probackup3 command is launched, except `help` and `version` commands. The time of the last log file creation is stored in `$BACKUP_PATH/log/log_rotation`. The zero value disables time-based rotation. Supported units: ms, s, min, h, d (min by default).

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`

`--compress-level=`*compression_level*

Defines the compression level. This option can 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`

`--compress`

Specifies the default compression algorithm and `--compress-level=1`. The default algorithm is selected among those supported by Postgres Pro according to the priorities: `zstd` (highest) -> `lz4` -> `zlib`. The `--compress` option overrides the `--compression-algorithm` and `--compress-level` settings and cannot be specified together with them.

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, remote mode 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.

`--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.

Buffer Options

You can use these options with all commands.

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

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

Remote Mode 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, see the section called “Configuring the Remote Mode” and the section called “Using pg_probackup3 in the Remote Mode”.

`--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 remote mode options.

`--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 Restore Options

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

`--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.

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

S3 Options

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

`--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.
- `vk` — VK Cloud storage. With this provider, the S3 host address `hb.vkcs.cloud`, port 443, and HTTPS protocol are only used. Custom values of the host, port, and protocol are ignored. The default value of region is `ru-msk`.

- `aws` — Amazon S3 storage, offered by Amazon Web Services (AWS). With this provider, the S3 host address *`bucket_name`*`.s3.`*`region`*`.amazonaws.com`, port 443, and HTTPS protocol are only used. Custom values of the host, port, and protocol are ignored. The default value of region is `us-east-1`.
- `google` —

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.

Once a pg_probackup3 command runs with the `--s3` option, pg_probackup3 starts running all commands that support parallel execution on 10 parallel threads (for details, see the section called "Running pg_probackup3 on Parallel Threads"). You can change the number of threads using the `-j`/`--threads` option.

`--s3-config-file=`*`path_to_config_file`*

Specifies the S3 configuration file. Settings in the configuration file override the environment variables. If this option is not specified, pg_probackup3 first looks for the S3 configuration file at `/etc/pg_probackup/s3.config` and then at `~postgres/.pg_probackup/s3.config`. The following is an example of the S3 configuration file:

```
access-key = ...
secret-key = ...
s3-host = localhost
s3-port = 9000
s3-bucket = s3demo
s3-region=us-east-1
s3-buffer-size = 32
s3-secure = on | https | http | off
```

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 versions earlier than 2.6.0. 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.

`--destroy-all-other-dbs`

By default, pg_probackup3 exits with an error if an attempt is made to perform a partial incremental restore since this destroys databases not included in the restore set. This flag allows you to suppress the error and proceed with the partial incremental restore (e.g., to keep a development database snapshot up-to-date with a production one). This option can be used with the restore command.

> **Important**
>
> Never use this flag in a production cluster.

`PGPROBACKUP_TESTS_SKIP_EMPTY_COMMIT`

Instructs pg_probackup3 to skip empty commits after pg_backup_stop.

Authors

Postgres Professional, Moscow, Russia.

Name

libpgprobackup — library with API to back up data, restore from backups, as well as validate and merge backups

Description

The libpgprobackup library contains functions to back up data, restore from backups, as well as validate and merge backups. The API provided enables you to create your own application to back up and restore data instead of using the command-line utility pg_probackup.

The library stores backups in files of its own format.

The library takes over the interaction with the database server, data processing, coding, and decoding, as well as creation and storing the metadata for backups.

An application that uses the library must provide it with access to the storage, such as a file system, S3 storage, or tape. However, file system operations, such as reading and storing backups, as well as storing the metadata, are the responsibility of the application. The application interacts with libpgprobackup as follows:

1. The application prepares the database instance for backing up and calls the libpgprobackup backup function.

2. libpgprobackup transforms the data files and WAL files to the pg_probackup3 format and redirects them to the application.

3. The client application sends the data to a file on the target storage (file system, S3 storage, or tape) and saves the backup metadata. Note that no intermediate storage is used.

4. To restore from a backup by calling the libpgprobackup restore function, to perform the backup integrity check by calling the validate function, or to merge backups by calling the merge function, the application provides the library with functions to get the backup data and metadata.

libpgprobackup is implemented in C++, but it can be used in applications written in different programming languages. The integration with C/C++ applications must be least complicated. The library uses the `extern "C"` calling convention.

libpgprobackup structures and functions are declared in the in the `probackup_lib.h` file.

Functions

The libpgprobackup library contains functions described below. When library functions are called, they accept structures with command parameters and a structure with pointers to functions that work with files and metadata.

bool `backup` (ConnectOptionsLib **conn_opt*, BackupOptionsLib **backup_opt*, MetadataProviderLib **metadata*);

Connects to the local or remote server and performs a backup. Returns `true` in the case of success and `false` in case of error.

Arguments:

- *`conn_opt`* — pointer to the ConnectOptionsLib structure that defines the options to connect to the Postgres Pro server, such as `pghost`, `pgdatabase`, `pgport`, `pguser`, `password`.

- *`backup_opt`* — pointer to the BackupOptionsLib structure that contains the backup options, such as the backup mode or number of threads.

- *metadata* — pointer to the MetadataProviderLib structure that provides callback functions for processing of backup metadata and file operations.

bool restore (RestoreOptionsLib **restore_opt*, MetadataProviderLib **metadata*);

Restores data from a backup using the parameters passed in the appropriate structures to the local file system. Returns true in the case of success and false in case of error.

Arguments:

- *restore_opt* — pointer to the RestoreOptionsLib structure that contains the options to restore from a backup.
- *metadata* — pointer to the MetadataProviderLib structure that provides callback functions for processing of backup metadata and file operations.

bool validate (RestoreOptionsLib **restore_opt*, MetadataProviderLib **metadata*, bool *withParents*);

Verifies that all the files required to restore the cluster from a backup are present and are not corrupt. If *withParents* is true, all the backups in the chain of parents are also validated. Returns true in the case of success and false in case of error.

Arguments:

- *restore_opt* — pointer to the RestoreOptionsLib structure that contains the options for integrity check of a backup.
- *metadata* — pointer to the MetadataProviderLib structure that provides callback functions for processing of backup metadata and file operations.
- *withParents* — a boolean value that defines whether parent backups must also be validated.

bool merge (MergeOptionsLib **merge_opt*, MetadataProviderLib **metadata*);

Merges a chain of incremental backups into one full backup. During the merge a new full backup created includes all the parent backups. If the parent backups do not have additional dependencies, they are removed after a successful merge. Returns true in the case of success and false in case of error.

Arguments:

- *merge_opt* — pointer to the MergeOptionsLib structure that contains the options for merging backups.
- *metadata* — pointer to the MetadataProviderLib structure that provides callback functions for processing of backup metadata and file operations.

uint64_t identify_system (ConnectOptionsLib **conn_opt*);

Returns the unique Postgres Pro server identifier. It is used for managing and restoring backups at the system level.

Arguments:

- *conn_opt* — pointer to the ConnectOptionsLib structure that defines the options to connect to the Postgres Pro server.

void set_probackup_logger (Logger *info*, Logger *warning*, Logger *error*);

Defines three callback logger functions that libpgprobackup will use to output information, warning, or error messages. Each of these functions must accept a string parameter with the message to be passed by the library.

Arguments:

- *info* — pointer to the function that will process information messages.
- *warning* — pointer to the function that will process warnings.
- *error* — pointer to the function that will process error messages.

Structures to Work with Files

The library gets feedback from the application through the structure of the MetadataProviderLib type, which must contain pointers to functions that work with files and metadata.

Callbacks from the library to functions passed by the application are used. Pointers to functions are passed to the library through the MetadataProviderLib structure. File read or write operations are defined in the CSource and Csink structures, respectively.

For the feedback from the application, library structures contain the `void *thisPointer` pointer, where the application can store the pointer to its own function or class instance. This pointer is passed to callback functions when they are called from the library.

MetadataProviderLib

A pointer to this structure is passed to all the main librobackup functions, such as backup, restore, validate, and merge. This structure defines methods to work with backup data, to write and read backup metadata, and to get the list of backups.

```
typedef struct
{
    CSink *(*get_sink_for_backup)(const char *backup_id, void
 *thisPointer);
    CSource *(*get_source_for_backup)(const char *backup_id, void
 *thisPointer);

    void (*register_backup)(PgproBackup *backup, void *ptr);
    PgproBackup *(*get_backup_by_id)(const char *backup_id, void
 *thisPointer);
    void (*free_backup)(PgproBackup *backup, void *thisPointer);
    char **(*list_backup_ids)(void *thisPointer);

    bool (*write_backup_status)(const char *backup_id, BackupStatus
 status,
                                void *thisPointer);

    void *thisPointer;
} MetadataProviderLib;
```

Where:

get_sink_for_backup

A pointer to the function that returns the pointer to the CSync structure (see CSource and Csink for more details). Needed for writing information into the backup. *backup_id* contains the string with the backup ID. In *thisPointer*, the application gets the pointer that was previously passed to the library through the structure.

get_source_for_backup

A pointer to the function that returns the pointer to the CSource structure (see CSource and Csink for more details). Needed for reading information from the backup. *backup_id* contains the

string with the backup ID. In *thisPointer*, the application gets the pointer that was previously passed to the library through the structure.

register_backup

A pointer to the function that stores the backup metadata. Accepts the PgproBackup structure.

get_backup_by_id

A pointer to the function that gets metadata of the backup defined by *backup_id*.

free_backup

Frees the memory for the PgproBackup structure returned by get_backup_by_id.

list_backup_ids

Returns the list of available backup IDs. The list contains pointers to strings with the zero character at the end. The last ponter in the list must also be zero. The memory for the list must be allocated by a C function such as malloc or strdup because the library frees the memory using the free function.

write_backup_status

A pointer to the function that saves the backup status. The backup status is updated separately from saving the rest of the backup metadata.

*void *thisPointer*

A pointer to be passed from the library to all callback functions.

CSource and Csink

These structures are needed for reading and writing data blocks to a backup file. Pointers to these functions must be returned by the get_source_for_backup and get_sink_for_backup functions, respectively. Each of these structures is actually a backup file that is open for reading or writing, respectively.

```
/* Support structure */
typedef struct
{
    unsigned char *ptr;
    size_t         len;
} c_buffer_t;

typedef struct
{
    c_buffer_t (*read_one)(void *thisPointer);
    ssize_t (*read)(char *buf, size_t size, void *thisPointer);
    void (*close)(void *thisPointer);
    void *thisPointer;
} CSource;
typedef struct
{
    /* Write one Pb structure. See @PbStructs. */
    size_t (*write_one)(uint8_t *buf, size_t size, void
 *thisPointer);
    ssize_t (*write)(char *buf, size_t size, void *thisPointer);
    /* Close this Sink. No more calls will be done. */
    void (*close)(void *thisPointer);
    void *thisPointer;
} CSink;
```

In the `thisPointer` field, an application can pass to the structure a pointer to a structure or application class that, for instance, contains a descriptor of an open file.

Functions from the CSource structure are used for reading the backup file, while functions from the CSink structure are used for writing the backup file. Each of these structures currently has two functions for that. The `read_one` and `write_one` functions perfrom lower-level operations with a data block. `write_one` saves the block size, while `read_one` first reads the block size and then the data block of this size.

`read` and `write` functions are used for simpler implementation of reading/writing. Their arguments are similar to those of common functions to work with files. Processing of the block size is done inside the library. These functions must return the size of the read/written data or 1 in case of error.

`close` functions must close the file.

PgproBackup

Backup metadata is passed in this structure.

```
typedef struct
{
    BackupStatus     backup_status;
    BackupMode       backup_mode;      /* Mode - one of
 BACKUP_MODE_xxx */
    char            *backup_id;        /* Identifier of the backup.
 */
    char            *parent_backup_id; /* Identifier of the parent
 backup. */
    BackupTimeLineID tli;       /* timeline of start and stop
 backup lsns */
    BackupXLogRecPtr start_lsn; /* backup's starting transaction
 log location */
    BackupXLogRecPtr stop_lsn; /* backup's finishing transaction
 log location */
    BackupTimestampTz   start_time; /* UTC time of backup creation
 */
    BackupTransactionId minxid;     /* min Xid for the moment of
 backup start */
    BackupMultiXactId
             minmulti; /* min multixact for the moment of backup
 start */

    bool stream; /* Was this backup taken in stream mode? I.e. does
 it include
                    all needed WAL files? */
    bool  from_replica;     /* Was this backup taken from replica
 */
    char *primary_conninfo; /* Connection parameters of the backup
 in the format
                               suitable for recovery.conf */
    char *note;

    /* For compatibility check */
    uint32_t block_size;
    uint32_t wal_block_size;
    char    *program_version;
    int      server_version;
```

```
   size_t uncompressed_bytes; ///< Size of data and non-data files
 before
                              ///< compression is applied
   size_t data_bytes; ///< Size of data and non-data files after
 compression is
                      ///< applied
   CompressAlg compress_alg;
   int         compress_level;

   BackupTimestampTz
           end_time; /* the moment when backup was finished, or
 the moment
                      * when we realized that backup is broken */
   BackupTimestampTz
           end_validation_time; /* UTC time when validation
 finished */

   BackupSource backup_source;/* direct, base or pro backup
 method*/

   size_t wal_bytes;// not used in pb3
} PgproBackup;
```

After execution of the backup function, the library calls a callback function register_backup, where the filled-in PgproBackup structure is passed. Software engineers can save the passed information at their discretion.

To get information on an existing backup, the library uses a callback function get_backup_by_id. From this function, the application must return a pointer to the PgproBackup structure with the metadata of the backup having the specified ID or a zero pointer in case of error.

To free the memory for the PgproBackup structure, the application must call the callback function free_backup.

Command Options

This section lists structures that are used to pass options to the commands. For more details, see the library header file `probackup_lib.h`.

connectOptionsLib

The following structure defines options to connect to the Postgres Pro server:

```
typedef struct connectOptionsLib
{
/* The database name. */
const char *pgdatabase;
/* Name of host to connect to. */
const char *pghost;
/* Port number to connect to at the server host, or socket file
 name
 * extension for Unix-domain connections.*/
const char *pgport;
/* Postgres Pro user name to connect as. */
const char *pguser;
/* Password to be used if the server demands password
 authentication. */
const char *password;
} ConnectOptionsLib;
```

backupOptionsLib

The structure below defines options to create a backup. Note that the `FULL` and `DELTA` backup modes are only supported so far.

```
typedef struct backupOptionsLib
{
    /* Number of threads, if backup mode supports multithreading */
    int num_threads;
    /* Backup mode PAGE, PTRACK, DELTA, AUTO, FULL*/
    BackupMode backup_mode;
    /* Backup source DIRECT, BASE or PRO */
    BackupSource backup_source;
    /* For DIRECT source is required to set up PGDATA. It is not
 required for
     * other sources */
    const char *pgdata;
    /* Backup Id if you wants to use custom id, otherwise it will
 be generated a
     * unique id using datetime of creation */
    const char *backup_id;
    /* Id of parent backup. It is not required for FULL backup mode
 */
    const char *parent_backup_id;
    /* Name of replication slot, if you use custom slot created by
     * pg_create_physical_replication_slot(). Otherwise will be
 used auto
     * generated slot */
    const char *replication_slot;
    bool        create_slot;
    const char *backup_note;
    /* If true, then WAL will be sent in stream mode, otherwise in
 archive mode
     */
    bool stream_wal;
    /* Progress flag. If true progress will be logged */
    bool        progress;
    const char *external_dir_str;
    /* Verify check sums. It is available only if checksums turn on
 on
     * PostgresPro server */
    bool verify_checksums;
    /* Compression algorithm, that is used for sending data between
 PostgresPro
     * server and libpgprobackup */
    CompressAlg compress_alg;
    /* Compression level, that is used for sending data between
 PostgresPro
     * server and libpgprobackup */
    int compress_level;
    /* Wait timeout for WAL segment archiving */
    uint32_t archive_timeout;
} BackupOptionsLib;
```

restoreOptionsLib

The following structure defines options to restore from a backup. They are also used for backup validation:

```
typedef struct restoreOptionsLib
{
    /* Number of threads */
    int         num_threads;
    /* Path to pgdata for restoration */
    const char *pgdata;
    /* Backup ID for restoration */
    char *backup_id;
    /* Progress flag. If true progress will be logged */
    bool        progress;
    /* Skip external directories */
    bool        skip_external_dirs;
    const char *external_mapping;
    const char *tablespace_mapping;
    /* Checksum page verification */
    bool        verify_checksums;
    /* Restore command to write in postgresql.auto.conf */
    /* https://postgrespro.ru/docs/enterprise/16/runtime-config-
wal#GUC-RESTORE-COMMAND */
    const char *config_content;
    /* Need recovery signal */
    bool need_recovery_signal;
    /* Need standby signal */
    bool need_standby_signal;
    /* No synchronization */
    bool no_sync;
} RestoreOptionsLib;
```

mergeOptionsLib

The following structure defines options to merge backups:

```
typedef struct mergeOptionsLib
{
    /* Number of threads */
    int         num_threads;
    /* Path to pgdata to restore to */
    const char *pgdata;
    /* Incremental backup ID for the merge */
    const char *backup_id;
    /* Target backup ID for the merge */
    const char *target_backup_id;
    /* Progress flag. If true progress will be logged */
    bool        progress;
    /* ID of the last increment */
    const char *merge_from_id;
    /* Time interval within which to merge backups */
    int            interval;
} MergeOptionsLib;
```

Constants

BackupMode

```
/*
```

```
 Backup Mode is how backup is taken.
 All DIFF modes require parent backup id passed in the
 BackupOptions.
 Parent backup id is ignored in the FULL mode.

 DIFF_AUTO returns selected mode in the metadata. Tentatively it
 prefers
 DIFF_PAGE if WAL summarization is available, if not it tries to do
 DIFF_PTRACK if ptrack is enabled and finally it falls back to
 DIFF_DELTA.
 In case when even DIFF_DELTA is not possible (no parent full
 backup exists)
 FULL backup is taken.
*/
typedef enum BackupMode
{
    BACKUP_MODE_INVALID = 0,
    BACKUP_MODE_DIFF_PAGE,   /* incremental page backup */
    BACKUP_MODE_DIFF_PTRACK, /* incremental page backup with ptrack
 system */
    BACKUP_MODE_DIFF_DELTA,  /* incremental page backup with lsn
 comparison */
    BACKUP_MODE_DIFF_AUTO, /* library selects diff backup mode
 automatically */
    BACKUP_MODE_FULL       /* full backup */
} BackupMode;
```

CompressAlg

```
/*
* Compression mode which is used to transfer data between PG server
 and the client lib.
* Default is NONE_COMPRESS.
*/
typedef enum CompressAlg
{
    NONE_COMPRESS = 0,
    ZLIB_COMPRESS,
    LZ4_COMPRESS,
    ZSTD_COMPRESS,
} CompressAlg;
```

BackupSource

```
/**
Backup Source is a method used by the client to access PGDATA.
*/
typedef enum
{
    /*
     * Direct access. The client reads data files directly.
     * Opens normal connection to execute PG_BACKUP_START/STOP.
     * Local file access. Multithreaded. No special PostgresPro
 edition
     * required.
     */
    BACKUP_SOURCE_DIRECT,
```

```
    /*
     * Uses pg_basebackup protocol.
     * Opens replication connection.
     * Remote file access. No special PostgresPro edition required.
     */
    BACKUP_SOURCE_BASE_BACKUP,
    /*
     * Uses pg_probackup protocol.
     * Opens replication connection.
     * Remote file access. Multithreaded. Only works with
 PostgresPro builds.
     * Supported PostgresPro versions start with ent-15.
     */
    BACKUP_SOURCE_PRO_BACKUP
} BackupSource;
```

Appendix A. Release Notes

Table of Contents

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, available exclusively in Postgres Pro Enterprise.
- Merging incremental backup chains: It is now possible to save disk space by merging chains of incremental backups.

Index

C

L

P