

Backup and Restore Overview



Copyright

© Postgres Professional, 2017, 2018, 2019.
Authors: Egor Rogov, Pavel Luzanov

Use of course materials

Non-commercial use of course materials (presentations, demonstrations) is permitted without restrictions. Commercial use is possible only with the written permission of Postgres Professional. Changes to course materials are prohibited.

Feedback

Send feedback, comments and suggestions to:
edu@postgrespro.ru

Denial of responsibility

In no event shall Postgres Professional be liable to any party for direct, indirect, special, incidental, or consequential damages, including lost profit, arising out of the use of course materials. Postgres Professional disclaims any warranties on course materials. Course materials are provided on an “as is” basis and Postgres Professional has no obligations to provide maintenance, support, updates, enhancements, or modifications.

Logical backup

Physical backup

What is logical backup

Backup of a table

Backup of a database

Backup of a cluster

SQL commands to restore data from scratch

- + backup of a separate object or a database is possible
- + restore is possible on another major version of PostgreSQL
- + restore is possible on another hardware or operating system
- poor performance

There are two types of backups, logical and physical.

A logical backup is a set of SQL commands that restores a cluster (or a database, or an individual object) from scratch.

Such a backup is essentially a plain text file, which gives a certain flexibility. For example, you can make a backup of only needed objects, you can edit the file to change the column's data types, etc.

In addition, SQL commands can be executed on another version of PostgreSQL (assuming SQL compatibility) or on a different hardware or operating system (that is, binary compatibility is not required).

However for a large database this is inefficient, since the execution of commands may take a lot of time. Also you can restore the system from a logical backup only to the time at which the backup was made.

<https://postgrespro.com/docs/postgresql/11/backup-dump>

COPY: table backup



Backup

outputs table (or arbitrary query) rows to a file, console, or program

Restore

adds rows from a file or console to an existing table

Server-side command

SQL command `COPY`

file must be available
to the postgres user
on the server

Client-side command

psql command `\COPY`

file must be available
to the user running psql
on the client

5

The `COPY` command can be used to make a backup of a single table.

The command allows to output a table (or an arbitrary query) rows either to a file, or to the console, or to the standard input of a program. You can specify a number of parameters, such as the format (text, csv or binary), field separator, null value text representation, etc.

The other variant of the same command reads lines from a file or from the console and writes them into a table. The table is not truncated, new rows are added to the existing ones.

The `COPY` command works much faster than similar `INSERT` commands, because there is no need in many roundtrips to the server, and the server does not parse the commands many times.

<https://postgrespro.com/docs/postgresql/11/sql-copy>

There is a client version of the `COPY` command with the same syntax. Unlike the server-side `COPY`, which is an SQL command, the client version is the `psql` command.

The file name specified in the SQL command corresponds to a file on the database server. The owner of PostgreSQL process (usually `postgres`) must have access to this file.

In the client version, the file is located on the client, and its content is transferred to the server.

<https://postgrespro.com/docs/postgresql/11/app-psql>

Backup

- outputs to a file or console either SQL script or an archive in a special format with a table of contents
- supports parallel execution
- allows to limit backup to certain tables or schemas, make it DML-only or DDL-only, etc

Restore

- SQL script by `psql`, special format by `pg_restore` (allows to limit the set of objects during restoration)
- supports parallel execution
- new database must be created from `template0`
- roles and tablespaces must be created in advance
- makes sense to collect statistics immediately after restore

To create a backup of a database, use the `pg_dump` utility. Depending on the specified parameters, the output is either an SQL script containing commands to re-create the selected objects, or a file in a special format with a table of contents.

To restore objects from an SQL script, just pass it to `psql`.

<https://postgrespro.com/docs/postgresql/11/app-pgdump>

To restore a backup in a special format, the `pg_restore` utility is required. It reads the file and converts it to regular `psql` commands. The advantage is that the set of objects can be limited not when creating a backup, but when restoring from it. In addition, backup and restore in the special format supports parallel execution.

<https://postgrespro.com/docs/postgresql/11/app-pgrestore>

The database to restore objects to must be created from `template0` database, since all changes made to `template1` will also get to backup. All the necessary roles and tablespaces must be created in advance, since they belong to the cluster as a whole, not to a particular database. After restoring the database, it makes sense to execute the `ANALYZE` command to collect statistics, which does not go to the backup.

Backup

backup of the entire cluster, including roles and tablespaces
outputs SQL script to a file or console
parallel execution is not supported, although it is possible to backup only global objects and then use `pg_dump` in parallel mode

Restore

using `psql`

Use the `pg_dumpall` utility to backup the entire cluster, including roles and tablespaces.

Since `pg_dumpall` requires access to all objects of all databases, it makes sense to run it on behalf of the superuser. The utility connects to each database of the cluster and run the `pg_dump` for it. It also backs up data related to the cluster as a whole.

The result of the `pg_dumpall` is a SQL script for `psql`. Other formats are not supported. This means that `pg_dumpall` does not support parallel execution, which can be a problem for large amounts of data. In this case, you can use the `--globals-only` key to backup only roles and tablespaces, and then backup each database manually using `pg_dump` in parallel mode.

<https://postgrespro.com/docs/postgresql/11/app-pg-dumpall>

What is physical backup

Cold and hot backup

Replication protocol

Standalone backups

Continuous archiving of WAL files

Based on recovery: copy of data + WAL files

- + good performance
- + point-in-time recovery is possible
- impossible to restore a separate database, only the entire cluster
- restore is possible only on the same major version and architecture

Physical backup uses the recovery mechanism based on write-ahead logging. This requires:

- copy of all cluster files (base backup),
- a set of WAL files needed to recover consistency.

If the data on disk is already consistent (this is the case if a copy was taken when the server was cleanly shutted down), then no logs are required.



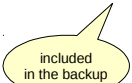

However, the presence of WAL files archive allows to restore the cluster to an arbitrary point in time. Thus, the cluster can be restored almost to the time of failure (or you can stop restoring at some prior point if needed).

High restore speed and the ability to create a backup on the fly (without shutting down the server) makes physical backups the main tool for periodic backups.

<https://postgrespro.com/docs/postgresql/11/backup-file>

<https://postgrespro.com/docs/postgresql/11/continuous-archiving>

Hot or cold?

| | Cold backup | | Hot backup |
|--|-------------------|--|--|
| Files are copied when the server is... | cleanly shut down | aborted (unclean shutdown)  | running  |
| WAL files are... | not needed | needed from the last checkpoint  | needed from the start of backup  |

10

Physical backup involves creating a copy of cluster files, that is contents of PGDATA directory plus all tablespaces created by the users.

If a backup is created when the server is shut down, it is called a **cold backup**. Such backup either contains consistent data (if the server was shutted down cleanly), or contains all the logs necessary for recovery (for example, if the server was aborted). This simplifies recovery, but requires server shutdown.

Note that some filesystems (like ZFS) can take snapshots of data. Such snapshot also can be used as a backup: from PostgreSQL perspective it looks just like an unclean shutdown.

If a backup is created when the server is running (which requires certain preparation: you cannot simply copy files), it is called a **hot backup**. In this case the procedure is a bit more complicated, but it is commonly used.

Files of a hot backup are known to contain inconsistent data. However, the recovery mechanism can be successfully applied to recover from a backup. This will require WAL files from the very start of a backup process. As backup can take quite a time, the server may recycle necessary WAL files, and hence these files must be archived.

Standalone backup contains both data files and WAL

Backup by the `pg_basebackup` utility

- connects to the server via replication protocol
- performs checkpoint
- switches to the next WAL file
- copies the data files to the specified directory
- switches to the next WAL file
- copies all WAL files generated between switches

Restore

- place standalone backup on a server
- start the database server

To create a hot backup, there is the `pg_basebackup` utility.

At the beginning, the utility performs a checkpoint and switches the log to a new file. Then a copy of the cluster data files is made, and then the log switches once again to a new file.

All WAL files between switches are also copied to the backup. The backup is called *standalone* because it contains everything you need to restore.

To restore, simply place the backup on a server and start PostgreSQL. Upon startup it will automatically restore consistency using existing WAL files and be ready to go.

<https://postgrespro.com/docs/postgresql/11/app-pgbasebackup>

The protocol

- transferring data and WAL stream
- backup and replication management commands

Served by the `wal_sender` process

The `wal_level` parameter set to `replica`

Replication slot

- server object through which the log records are received
- remembers which record were sent last
- WAL file is not deleted until it is completely sent and acknowledged

In order to save all the WAL files required for recovery, generated by the server during the file copying, the utility connects to the server using a special replication protocol. Despite the name, this protocol is used not only for replication (which will be discussed in the next lesson), but also for backup. The protocol allows to receive a stream of log records in parallel with file copying (which, in fact, is also done using the replication protocol).

To prevent the server from deleting the necessary WAL files prematurely, a replication slot can be used.

In order for the connection to be possible, a number of settings are needed.

First, the role must have the `REPLICATION` attribute (or be a superuser), and connection must be allowed in the `pg_hba.conf` configuration file.

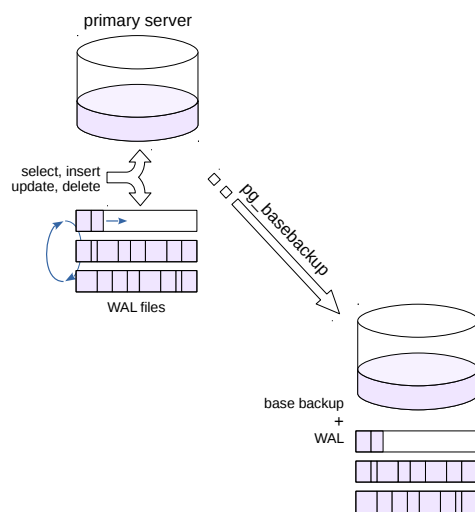
Second, the `max_wal_senders` parameter should be set to a sufficiently large value. This parameter limits the number of simultaneously running `wal_sender` processes servicing replication protocol connections.

Third, the `wal_level` parameter, which determines the amount of information in the log, must be set to the `replica` value.

Starting from version 10, the default settings already satisfies all these requirements (for local connection).

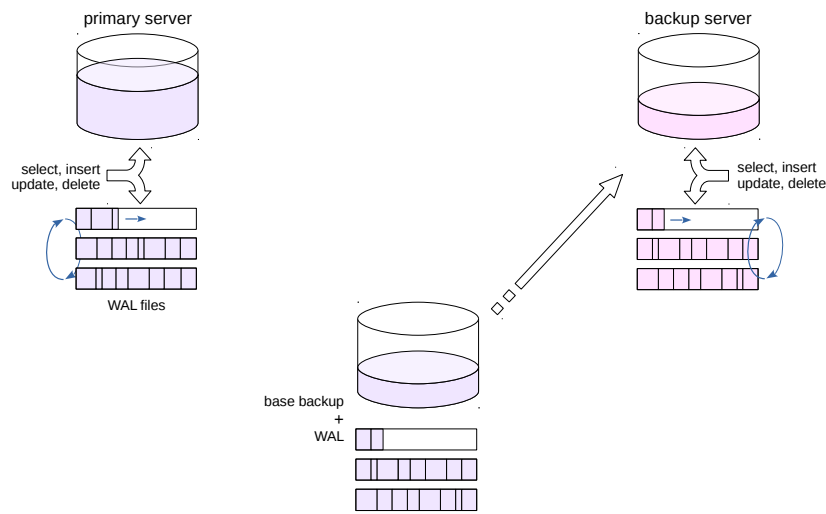
<https://postgrespro.com/docs/postgresql/11/protocol-replication>

Standalone backup



On this illustration the primary server handles incoming requests. This creates WAL records and changes the state of the databases (first in the buffer cache, then on disk). WAL files are overwritten cyclically (more precisely, the old segments are deleted, since the file names are unique). Somewhere (usually on another physical server) a backup is created: a base backup plus a set of WAL files.

Restore



When restoring, a base backup including all the necessary WAL files is copied to another server (shown on the right).

After starting the server, it automatically restores consistency and gets to work. Recovery occurs to the point in which the backup was made.

Of course in the meantime the primary server can go far ahead from this point. Hence, in case of a primary server failure some data will be lost.

Continuous archiving (file-based archive)

- WAL files are copied to the archive upon completion
- archiving is managed by the server
- inevitable delays in getting data to the archive

Stream archive

- stream of log records is constantly written to the archive
- external utility is required
- delays are minimal

Further development of the idea of a hot backup: since we have a copy of the data files and WAL files, we are able to restore the system to an arbitrary moment simply by constantly saving new log records generated by the primary server.

There are two ways to do this. The first is to archive WAL files before the server deletes them. This can be implemented by special server settings. Unfortunately, with this option the WAL file will not be archived until the server switches to another file. Each WAL file is 16 MB, so under low workload it may take some time.

The second way is to get the stream of log records by the replication protocol and write it to the archive. With this option, the delays will be minimal, but it requires running (and monitoring) a separate utility to receive data from the stream.

The archiver process

Parameters

archive_mode = on

archive_command a shell command to copy completed WAL file to the archive

archive_timeout maximum time to switch to the new WAL file

Algorithm

when a WAL file is completed, the *archive_command* command is called
if the command ends with status 0, the WAL file is deleted

otherwise (in particular, if the command is not specified), the WAL file remains intact until the subsequent attempt is successful

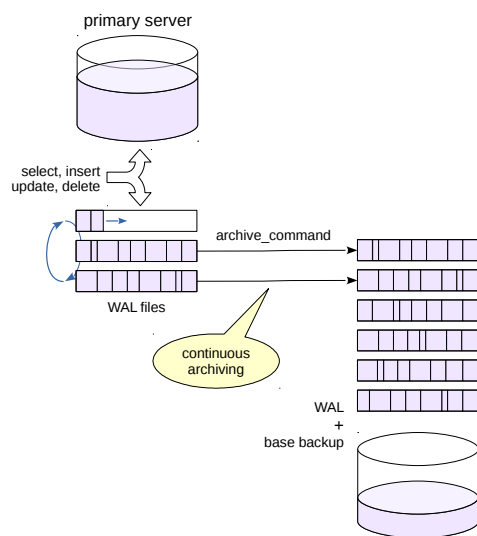
Continuous archiving is served by the archiver background process.

PostgreSQL allows to specify an arbitrary shell command in the *archive_command* parameter, which will copy a completed WAL file to the archive. The archiving is enabled by the parameter *archive_mode* = on. The *archive_timeout* parameter allows to specify the maximum time to switch to a new WAL file, to guarantee that the log files are archived at least once a specified period of time. Switching to a new file can also be done manually using the `pg_switch_log()` function.

The general algorithm is as follows. When the next WAL file is completed, the *archive_command* is called. If it ends with a zero status, then the file is known to be copied and can be deleted. If not, then the file (and files following it) will not be deleted, and the server will periodically try to execute the command until it receives 0.

<https://postgrespro.com/docs/postgresql/11/continuous-archiving>

Continuous archiving



This figure shows the primary server configured for continuous archiving. The completed WAL files are copied to the archive using the command in the *archive_command* parameter. Usually the archive is located on another physical server to increase fault tolerance.

There is also a base backup (usually a set of base backups made at different times).

The `pg_receivewal` utility

connects via replication protocol (slot can be used)
and writes stream of WAL records to files

starting position is the beginning of the WAL file
following the last completed file found in the directory,
or the beginning of the current server WAL file in case the directory is empty

in contrast to the continuous archiving, records are written without delay

when switching over to a new server, the `pg_receivewal` utility
must be re-run with new parameters

Another solution is to use the `pg_receivewal` utility to write WAL records to the archive via replication protocol (before version 10, this utility was called `pg_receivexlog`).

Usually, the utility runs on a separate archive server and connects to the primary server with the command line parameters. The utility can (and should) use the replication slot to ensure that it does not lose records.

The utility generates files in the same way as the server does, and writes them to the specified directory. Not yet completed files are marked with the `.partial` prefix.

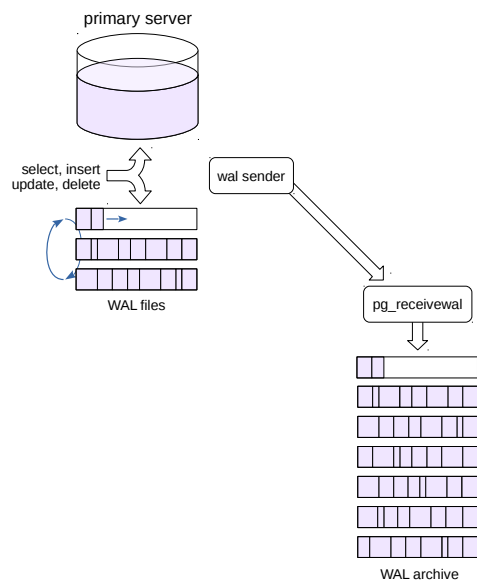
Streaming always starts from the beginning of a file, following the last completed file found in the archive directory. If the archive is empty (first start), archiving starts from the beginning of the current WAL file. This ensures that there will be no "holes" in the archive.

When switching over to a new server, the utility must be stopped and restarted with the appropriate parameters.

It is necessary to take into account that the utility itself does not start automatically (as a service) and is not demonized.

<https://postgrespro.com/docs/postgresql/11/app-pgreceivewal>

Streaming archive



The `pg_receivewal` utility connects to the server via streaming replication protocol. The connection is processed by a separate `walsender` process (this must be taken into account when setting the `max_wal_senders` parameter).

The utility gets data without waiting for the file to complete.

Assumes WAL archiving is set up

Backup by the `pg_basebackup` utility

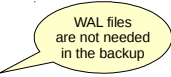
- connects to the server via replication protocol

- performs checkpoint

- switches to the next WAL file

- copies the data files to the specified directory

- switches to the next WAL file



WAL files
are not needed
in the backup

Restore

- place standalone backup on a server

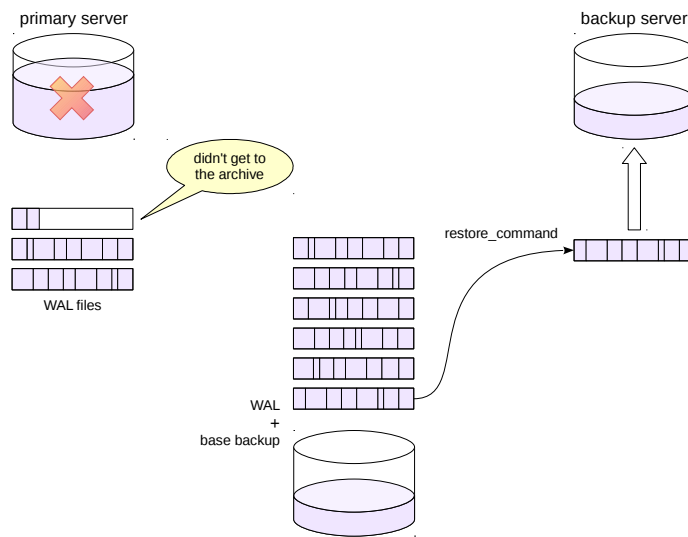
- create the `recovery.conf` file (parameters to read WAL files from the archive, specify recovery target point)

- start the database server

To create a backup with configured continuous archiving, the same utility `pg_basebackup` is used, only with a different set of parameters. The only difference is that WAL files are not saved to the backup, as they already present in the archive.

Recovery in this case is a bit more complicated. In addition to copying the backup, you need to create the `recovery.conf` file that controls the recovery. This file contains the `restore_command` (it is like `archive_command`, but copies the necessary files from the archive back to the server) and the recovery target point.

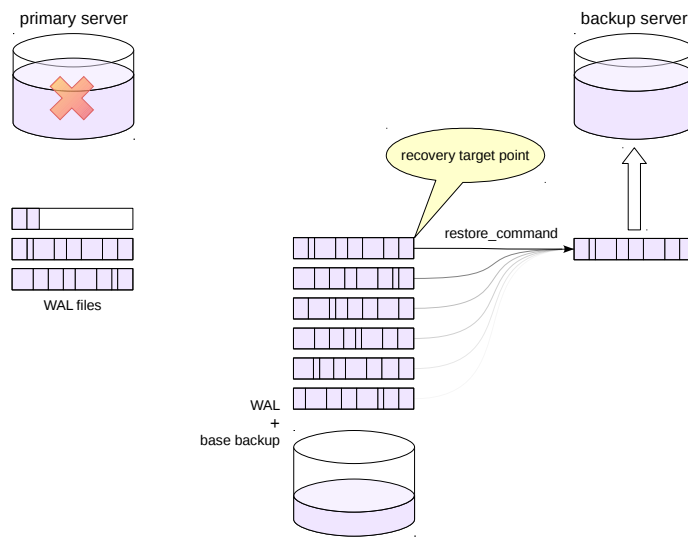
Restore



The restore procedure (for example, when the primary server fails) is as follows. The base backup is copied to some other (or the same) server and the recovery .conf control file is created. The PostgreSQL server starts and begins reading WAL files from the archive (using `restore_command`) and applying WAL records.

Please note that the last non-completed WAL file from the primary server is not archived in case of continuous archiving. However, the file can be manually copied to the backup server in the `pg_wal` directory, if the file is available. Of course, there may be several such files, but only as a result of some kind of failure during archiving.

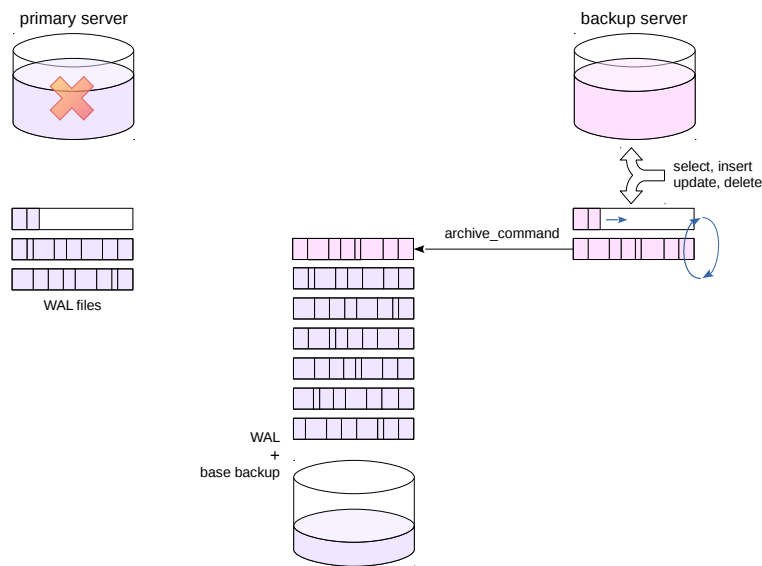
Restore



After reading all the WAL files from the archive (and all the WAL files from the `pg_wal` directory), the backup server is up-to-date. The maximum possible data loss is an non-completed WAL file that did not managed to get to the archive.

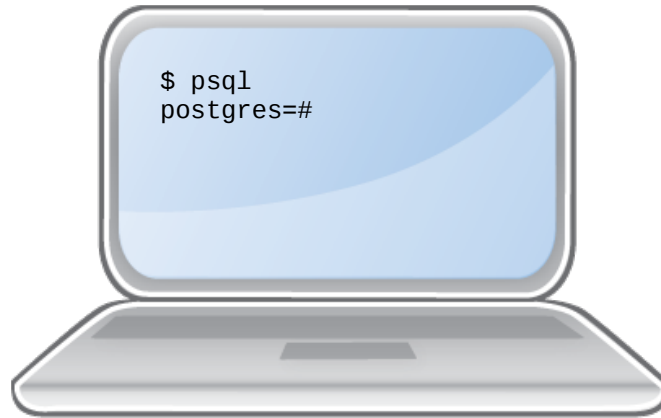
Specifying the recovery target point in the `recovery.conf` file allows to stop applying the log records at the desired point in time.

Restore



After that, the backup server goes into normal operation, accepts requests, writes its own WAL files to the archive, and so on, acting as a new full-fledged primary server.

If the new server is supposed to be used instead of the old primary server, then it makes sense to locate it on a comparable hardware, in order to avoid performance degradation.



Logical backup: SQL commands for restoring objects

COPY command, pg_dump and pg_dumpall utilities

Physical backup: copy of data + WAL files

pg_basebackup utility

WAL archive

file-based or streaming

allows for point-in-time recovery

1. Create a database and a table in it with several rows.
2. Make a logical copy of the database using the `pg_dump` utility.
3. Drop the database and restore it from the backup.
4. Make a standalone physical backup of the cluster using the `pg_basebackup` utility.
5. Make some changes to the table.
6. Restore the new cluster from the physical backup on Beta server and check that the database does not contain later changes.